Report

v. 1.0

Customer

Eco

Smart Contract Audit

Eco. Phase I

5th November 2022

Report prepared by

ABDK Consulting

Contents

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 66b85619 commit and provided a list of issues found in the code.

Files with the indicated issues were re-audited during Phase II.

3 Project scope

Repositories:

- Original Repository

Files:

/			
	Migrations.sol		
clone/			
	CloneFactory.sol		
deploy/			
	EcoBootstrap.sol	EcoFaucet.sol	EcoInitializable.sol
	EcoTestCleanup.sol	EcoTokenInit.sol	
governance/			
	CurrencyTimer.sol	IGeneration.sol	CurrencyGovernance.sol
	TimedPolicies.sol	ECOxLockup.sol	VotingPower.sol
	ILockups.sol	Inflation.sol	ITimeNotifier.sol
	Lockup.sol	PolicyProposals.sol	PolicyVotes.sol
	Proposal.sol	SimplePolicySetter.sol	Single TrusteeReplace-ment.propo.sol
	TrustedNodes.sol	Trustee Replacement.propo.sol	
Policy/			
	ERC1820Client.sol	Policed.sol	PolicedUtils.sol
	Policy.sol	PolicyInit.sol	
proxy/			
	ForwardProxy.sol	ForwardTarget.sol	
utils/			
	TimeUtils.sol		
VDF/			
	BigNumber.sol	IsPrime.sol	VDFVerifier.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 4 critical, 14 major, and a few less important issues. All identified Critical issues have been fixed.

Issues	Active	Fixed
Severity		
Critical	**0**	**4**
Major	**3**	**11**
Moderate	**3**	**18**
Minor	**34**	**242**

Fixed 275 out of 315 issues

6 Critical Issues

CVF-1.1. FIXED

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

Description There is no way to challenge the “totalSum” value for a proposal.

Recommendation Consider adding the following code into the “respondToChallenge” function: if (_index == proposal.amountOfAccounts - 1) { require (_sum + _claimedBalance == proposal.totalSum, ”...”); }

```
44 uint256 totalSum;
```

CVF-1.2. FIXED

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

Description As it is not checked that the challenged address is nonzero, one can populate the tree with leafs of the zero account and assigning them even indices. Then every non-zero account will have neighbors with zero address and can’t verify their balances. Moreover these zero account leafs may contain nonzero sum or even nonzero balances.

```
420 proposal.challengeResponses[_index].account = _account;
```

CVF-1.3. FIXED

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

Description The “index” value store inside a leaf is not used to determine the leaf’s position in the tree, so index uniqueness across leafs is not guarantees. This allows a proposer to pack all the valid accounts into a tree and add more leafs with valid indexes but invalid data. Such tree could stand all the challenges and thus would most probably be accepted, but them invalid leafs from it could be claimed in parallel with valid ones.

Recommendation Consider using a accoutn index as the positions of the correponding tree leaf, so index bits would be used to to determine the whether to go left or right at each step when traversing from the tree root to the leaf.

673
```
if (computedHash < proofElement) {
```

CVF-1.4. FIXED

- **Category** Flaw
- **Source** ECO.sol

Description This could make voting power distribution inconsistent with actual token balances and delegates leading to very messy things. Lets consider the following scenario: 1. Alice and Bob both delegated to the same address Carol 2. Alice has 100 tokens, while Bob and Carol have zero tokens, so Carol’s voting power is 100 3. Bob receives 100 tokens from a locked address, so Bob now has 100 tokens, but Carol’s voting power didn’t change and remains 100 4. Bob, whose address is not locked, sends his 100 tokens to a non-locked address, so Carol’s voting power is reduced by 100 and is now zero 5. Alice, whose address is also not locked, tries to send here tokens to a non-locked address, but this attempt fails as it would make Carol’s voting power negative

Recommendation Consider revising the logic.

60
```
// If to or from is a lockup early return so voting power and
   ↪ delegation remain
```

7 Major Issues

CVF-1.5. INFO

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

Recommendation The security of Pietrzak VDF was proven assuming that N is a product of safe primes (N=pq where p and q are of form 2r+1 for prime r), which is unknown for the N in factoring challenges.

Client Comment *This N has seen use in several VDF implementations.*

```
20 bytes public constant N =
```

CVF-1.6. FIXED

- **Category** Flaw
- **Source** VDFVerifier.sol

Description The value “10” here is too small, as it means that an attacker needs to try only about 1M non-prime values in a few seconds to bypass this check. If an attacker can choose ‘x‘, this would be a critical bug. It would then be possible to shortcut the VDF computation.

```
93 require(isProbablePrime(_x, 10), "x must be probable prime");
```

CVF-1.7. INFO

- **Category** Flaw
- **Source** VDFVerifier.sol

Description Since the VDF evaluation does not depend on the sender, a VDF proof can be stolen or frontrunned, so that an adversary submits the full proof before the VDf evaluator and sets the mapping.

Client Comment *The proof cannot be stolen because it is only done once. There is no reward for submitting the proof, so someone submitting a valid proof instead is not unfair, and still allows the system to progress.*

```
177 verified[keccak256(abi.encode(s.t, s.x))] = keccak256(
        s.y.asBytes(nlen)
    );
180 delete (state[msg.sender]);
```

CVF-1.8. FIXED

- **Category** Suboptimal
- **Source** Inflation.sol

Description Running over all numbers in the interval is expensive but not necessary. Caller should provide a small z (say<1000) such that (blockhash+z) is prime. It won't be possible to exploit that several primes are in the interval (on average 1 of 177 for 256-it numbers). It is possible to allow hints for several latest block numbers instead of one, and they all can be packed in a single 256bit value.

```
149 while (
150     ((x % 3) == 0) ||
      (x % 5 == 0) ||
      (x % 7 == 0) ||
      !vdfVerifier.isProbablePrime(x, 10)
  ) {
      x = x + 2;
```

CVF-1.9. FIXED

- **Category** Flaw
- **Source** IsPrime.sol

Description As long as the hash of the previous block is used for random number generation, and the number "_n" comes from the user, an attacker, that already knows the hash of the previous block, could find a composite value that will stand the check. The attacker would need to find such number before the next block is mined, i.e. within a few seconds. In order to do so, the attacker would need to try 4^k numbers. It is known that the composite numbers that achieve this bound exist and follow a special form n = (2q+1)(4q+1) for prime q. Such numbers can be constructed in advance. Worse enough, these pseudoprimes can be tested in parallel, which takes fractions of second on decent hardware or in a cloud. For k = 10 this means about 1M numbers which is doable.

Recommendation Consider implementing a schema where the number to be checked is committed to the blockchain before the block, whose hash is used for generating random numbers, is mined.

```
60 abi.encode(blockhash(block.number - 1), i)
```

CVF-1.10. FIXED

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

Recommendation It seems that the "innerAdd" function only require that the length of "max" is not less than the length of "min" so full comparison is redundant here and it would be enough to just compare the arguments lengths.

```
263 int256 compare = cmp(_a, _b);
```

CVF-1.11. INFO

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

Description This number includes all the challenges, even successfully answered and those, that invalid, but not yet answered. So, in case when an invalid root hash was proposed and challenged. one may submit many duplicate or invalid challenges from many addresses to get the majority of the proposer's stake. This is unfair.

Recommendation Consider transferring the whole stake to the first successful challenger.

Client Comment *This suggestion is a bad incentive. The intent of the reward is to cover the gas costs of challenging, not as a money-making opportunity. The suggested change would disincentivize anyone to start challenging because there would be almost guaranteed no reward for doing so.*

```
600 proposal.totalChallenges;
```

CVF-1.12. FIXED

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

Recommendation Should be ">=".

```
211 if (blockNumber > ckpts[ckpts.length - 1].fromBlock)
```

CVF-1.13. FIXED

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

Description The "TimedPolicies(_self).getNotificationHashesLength()" expression is calculated on every loop iteration.

Recommendation Consider calculating once and reusing.

```
73 i < TimedPolicies(_self).getNotificationHashesLength();
```

CVF-1.14. FIXED

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

Description The "_self" contract is called here on every loop iteration.

Recommendation Consider implementing an ability to obtain all notification hashes from it in one call.

```
76 notificationHashes.push(TimedPolicies(_self).notificationHashes(i));
```

CVF-1.15. FIXED

- **Category** Flaw
- **Source** TrustedNodes.sol

Description The returned value is ignored.

Recommendation Consider checking the returned value or adding a comment explaining why such check is unnecessary here.

```
116 ECOx(policyFor(ID_ECOX)).transfer(msg.sender, _reward);
```

CVF-1.16. FIXED

- **Category** Suboptimal
- **Source** PolicyInit.sol

Description These values are calculated _tokenResolvers.length + 1 times for every index.

Recommendation Consider allocating an in-memory array for these values, calculating all of them once, and then using value from the in-memory array in the main two-dimensional loop.

Client Comment *This code is unused and unneeded, so has been removed*

```
51 PolicedUtils a = PolicedUtils(
       ERC1820REGISTRY.getInterfaceImplementer(
           address(this),
           _tokenResolvers[i]
       )
   );
```

```
58     PolicedUtils b = PolicedUtils(
           ERC1820REGISTRY.getInterfaceImplementer(
60             address(this),
               _tokenResolvers[j]
           )
       );
```

CVF-1.17. FIXED

- **Category** Flaw
- **Source** ForwardTarget.sol

Description It is not checked that the implementation address set here is non-zero, thus it is not guaranteed that this function cannot be called again.

Recommendation Consider adding a “require” statement to check that the implementation address is not zero.

```
45 setImplementation(address(ForwardTarget(_self).implementation()));
```

CVF-1.18. FIXED

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

Recommendation It would be more efficient to accept the hash of an interface label instead of the interface label itself, as in many cases interface labels are constant strings and their hashes could be precomputed.

Client Comment *These functions are unused and this file was sourced from https://github.com/0xjac/ERC1820/blob/master/contracts/ERC1820Client.sol which is a github repo used in the original EIP. Both functions are superceeded with ones that use the hash and so these functions are removed.*

```
13 function interfaceAddr(address _addr, string memory _interfaceLabel)
21             keccak256(abi.encodePacked(_interfaceLabel))
27     string memory _interfaceLabel,
32         keccak256(abi.encodePacked(_interfaceLabel)),
```

8 Moderate Issues

CVF-1.19. FIXED

- **Category** Procedural
- **Source** VDFVerifier.sol

Recommendation It should be mentioned explicitly in the documentation that this contract is a secure VDF verifier only under the assumption that input ‘x‘ is not controlled by an attacker. This is not limited to the primality test (see in another comment) but also to the problem of uniqueness and amortization. For example, if one finds x such that x^2 passes the primality test, it is then possible to ”prove” a VDF for x^2 → y^2 out of x→y. Consider explicitly documenting the requirements to x.

Client Comment *Finding a valid prime x such that x^2 is also a false positive to the primality test is too hard of a contstraint on the attacker. The value of x is controlled by a potential attacker, but is highly limited (must be close after the previous blockhash, caste as a uint256). This does not guarantee that any such malicious x is valid.*

```
10 contract VDFVerifier is PolicedUtils, IsPrime {
```

CVF-1.20. FIXED

- **Category** Flaw
- **Source** VDFVerifier.sol

Description The “byteLength” function returns a value rounded up to a multiple of 32, so this check basically ensures that y is at least 257 bits long.

Recommendation In order to check that a value is at least 512 bits long one need to check that “byteLength” or the value is at least 64, and in case it is exactly 64, additionally check that the highest bit is set to one.

```
87 require(
       y.byteLength() >= 64,
       "The secret (y) must be at least 512 bit long"
90 );
```

CVF-1.21. FIXED

- **Category** Documentation
- **Source** Inflation.sol

Description It is not true, as this function could be called at any time before the “submitEntropyVDF” function was called.

```
88 * Can only be called after all pay-outs
   * have been claimed.
```

CVF-1.22. FIXED

- **Category** Procedural
- **Source** Inflation.sol

Description The returned value is ignored.

Recommendation Consider checking the returned value.

```
106 getToken().transfer(
```

```
259 getToken().transfer(_who, prize);
```

CVF-1.23. FIXED

- **Category** Procedural
- **Source** Inflation.sol

Recommendation Returned value is ignored.

```
259 getToken().transfer(_who, prize);
```

CVF-1.24. FIXED

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

Description This formula is fine for real division, while the actual code uses integer division with rounding down. This makes the formula incorrect. Here is an example: x = 7, N = 5 2^((x - 2) / 2) ~ 5.657 > N 2^floor((x - 2) / 2) ~ 4 < N

Recommendation Consider using the following formula instead: 2^(x - 2) < N^2

185
```
            2 ^ ((x - 2)/2) < N
```

189
```
2**((requestsByChallenger - 2) / 2) < proposal.amountOfAccounts,
```

CVF-1.25. INFO

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

Description Here the “challengeEnds” value is updated for a challenge, but the “lastLiveChallenge” value is not updated for the corresponding proposal.

Recommendation Consider updating the “lastLiveChallenge” value as well.

496
```
challenge.challengeEnds = challenge.challengeEnds + 1 hours;
```

CVF-1.26. INFO

- **Category** Readability
- **Source** InflationRootHashProposal.sol

Recommendation Should be: else if (proposal.status == RootHashStatus.Pending) {

Client Comment *previous conditional is 'acceptedRootHash == 0 && getTime() > proposal.newChallengerSubmissionEnds && getTime() > proposal.lastLiveChallenge' and can fail for reasons other than 'acceptedRootHash != 0' this would be a functional change and therefore readability is maintained in a different way.*

```
523 if (
        acceptedRootHash != 0 && proposal.status == RootHashStatus.
            ↪ Pending
    ) {
```

CVF-1.27. FIXED

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

Description The returned value is ignored. An unsuccessful transfer could be treated as a successful one.

Recommendation Consider always checking the returned value or using the "SafeERC20" library.

```
587     getToken().transfer(_who, proposal.stakedAmount);
```

```
601     getToken().transfer(_who, amount);
```

```
625 getToken().transfer(
        address(uint160(policy)),
        getToken().balanceOf(address(this))
    );
```

```
713 getToken().transferFrom(_submitter, address(this), _fee);
```

CVF-1.28. FIXED

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

Description Overflow is possible here.

Recommendation Consider using a safe version of the left shift operation.

```
67 uint256 _preciseRatio = (_ecoXValue << PRECISION) / initialSupply;
```

CVF-1.29. FIXED

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

Description The "_requiredStake" value could be zero in case the "totalStake" value is 1.

Recommendation Consider using the "totalStake == 0" condition instead.

```
166 } else if (_requiredStake == 0) {
        // Nobody voted
        _res = Result.Failed;
```

CVF-1.30. FIXED

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

Recommendation Probably should be '<'.

```
169 } else if (yesStake <= _requiredStake) {
```

CVF-1.31. FIXED

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

Description Return value is ignored.

```
181 getToken().transfer(
```

CVF-1.32. FIXED

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

Description This function allows modifying the name and the symbol of a deployed token. Usually, token metadata is meant to be immutable and application may not be ready to handle metadata changes.

Recommendation Consider keeping token metadata immutable.

```
74 function copyTokenMetadata(address _target) internal {
```

CVF-1.33. INFO

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

Description This copies the current “internalGenerator” value from the “_self” contract, not the initial value. Note, that the current “internalGeneration” value of the original contract instant could be changed by anybody by calling the unprotected “incrementGeneration” function.

Recommendation Consider forbidding calling the “incrementGeneration” function of the main contract instant or setting the “internalGeneration” value here to “GENERATION_START”.

Client Comment *The initialize function on TimedPolicies is not used because TimedPolicies is frequently cloned but instead used for setting up the proxy. Therefore the main contract instance is the one that is used in day to day functionality. Copying the value in this way allows a process to make a short lived clone of TimedPolicies and have the generation value persist to the copy.*

```
70 internalGeneration = TimedPolicies(_self).internalGeneration();
```

CVF-1.34. FIXED

- **Category** Flaw
- **Source** Lockup.sol

Description This function reverts in case the contract's balance exceeds the total deposits amount adjusted by the interest.

Recommendation Consider returning zero in such a case. With the current implementation, a maliciour user may sent extra tokens to the contract to make this funcgtion revert, thus disrupting other contracts that call this function.

Client Comment *This function was removed as the lockups were changed to no longer hold the rewards but instead calculated and minted/burned the reward/penalty amounts as needed.*

```
91 function mintNeeded() external view returns (uint256) {
```

CVF-1.35. FIXED

- **Category** Flaw
- **Source** Lockup.sol

Description The returned value is ignored.

```
120 getToken().transfer(_owner, _amount);
```

```
137 getToken().transferFrom(_payer, address(this), _amount);
```

CVF-1.36. FIXED

- **Category** Flaw
- **Source** CurrencyGovernance.sol

Description This function is not idempotent, i.e. calling it several times in a row doesn't necessary has the same result as calling it once.

Recommendation Consider modifying the function to make it idempotent. The simplest way to do so is to remove "else" keywords from it.

```
72 function updateStage() public {
```

CVF-1.37. FIXED

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

Description The returned value is ignored.

```
288 getToken().transfer(receiver, REFUND_IF_LOST);
```

```
305 getToken().transfer(
```

CVF-1.38. FIXED

- **Category** Flaw
- **Source** ECOxLockup.sol

Description This check will fail in case the "_destination" address never voted, but the current generation is less than 2.

Recommendation Consider performing the check only in case the "votingTracker[_destination]" value is non-zero.

Client Comment *The value for generation is initialized at 1000. However, a comment has been added for clarity.*

```
56 require(
       votingTracker[_destination] < currentGeneration - 1,
       "Must not vote in the generation on or before withdrawing"
   );
```

CVF-1.39. FIXED

- **Category** Procedural
- **Source** ECOxLockup.sol

Description The returned value is ignored.

```
63 getToken().transfer(_destination, _amount);
```

9 Minor Issues

CVF-1.40. FIXED

- **Category** Procedural
- **Source** VDFVerifier.sol

Recommendation Should be "^0.8.0" unless there is something special about this particular version. Also relevant for the next files: Inflation.sol, IsPrime.sol, BigNumber.sol, InflationRootHashProposal.sol, ECOx.sol, EcoBalanceStore.sol, InflationCheckpoints.sol, PolicyVotes.sol, TokenEvents.sol, VoteCheckpoints.sol, IEcoBalanceStoreGenerationBalance.sol, ERC20.sol, TimedPolicies.sol, ECO.sol, TrustedNodes.sol, SimplePolicySetter.sol, Lockup.sol, CurrencyGovernance.sol, PolicyProposals.sol, CurrencyTimer.sol, ECOxLockup.sol, TrusteeReplacement.propo.sol, Policed.sol, EcoTestCleanup.sol, EcoInitializable.sol, EcoFaucet.sol, EcoBootstrap.sol, ForwardProxy.sol, PolicyInit.sol, ForwardTarget.sol, ERC1820Client.sol, Proposal.sol, PolicedUtils.sol, CloneFactory.sol, SingleTrusteeReplacement.propo.sol, EcoTokenInit.sol, Migrations.sol, VotingPower.sol, ILockups.sol, TimeUtils.sol, IGeneration.sol, Policy.sol, ITimeNotifier.sol.

```
2 pragma solidity ^0.8.9;
```

CVF-1.41. FIXED

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

Recommendation It would be more efficient to change the type of this variable to "bytes32 [8]".

```
20 bytes public constant N =
```

CVF-1.42. FIXED

- **Category** Readability
- **Source** VDFVerifier.sol

Recommendation Duplicate "state is".

```
24 * from a single verifier. Once verification is complete, state is
   * state is removed, and (if succesfully verified) replaced by a
     ↪ entry
```

CVF-1.43. FIXED

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

Description This variable is never read.

Recommendation Consider removing it.

```
45 address private destroyer;
```

CVF-1.44. FIXED

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

Recommendation Events are usually named via nouns, such as “SuccessfulVerification” or just “Success”.or “Verification”.

```
49 event Verified(uint256 x, uint256 t, bytes y);
```

CVF-1.45. FIXED

- **Category** Procedural
- **Source** VDFVerifier.sol

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

```
55 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.46. INFO

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

Description The contract could be cloned from the outside. This function is redundant. Calling this function on a clone would clone a proxy rather than the original implementation.

Recommendation Consider removing this function.

Client Comment *Cloning directly from a template is preferred in our system. Cloning of a proxy is disallowed, so is not an issue.*

```
58 function clone() public override returns (address) {
```

CVF-1.47. INFO

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

Description This function doesn't check whether the solution was already verified.

Recommendation Consider adding such check and reverting in case it was.

Client Comment *Our system only cares about recieving one specific solution to the VDF. Re-verifying has no effect, positive or negative.*

```
74 function start(
```

CVF-1.48. FIXED

- **Category** Procedural
- **Source** VDFVerifier.sol

Recommendation These checks should be performed at the very beginning of the function.

```
84 require(_t >= 2, "t must be at least 2");
   require(_x > 1, "The commitment (x) must be > 1");
```

CVF-1.49. FIXED

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

Recommendation The value “64” should be a named constant.

```
88 y.byteLength() >= 64,
```

CVF-1.50. FIXED

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

Recommendation The value “10” should be a named constant.

```
93 require(isProbablePrime(_x, 10), "x must be probable prime");
```

CVF-1.51. FIXED

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

Description The expression “state[msg.sender]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
95 state[msg.sender].progress = 0; // reset the contract
   state[msg.sender].t = _t;
```

```
98 state[msg.sender].x = _x;
   state[msg.sender].y = y;
```

```
101 state[msg.sender].xi = x2; // our time-lock-puzzle is for x2 = x^2;
       ↪ x2 is a QR mod n
    state[msg.sender].yi = y;
```

CVF-1.52. FIXED

- **Category** Readability
- **Source** VDFVerifier.sol

Recommendation typo: ”complete”

```
111 * competed the verification successfully.
```

CVF-1.53. FIXED

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

Description This condition makes the ‘nextProgress‘ parameter redundant.

Recommendation Consider removing it.

```
119 _nextProgress == s.progress + 1 && s.x > 0,
```

CVF-1.54. FIXED

- **Category** Procedural
- **Source** VDFVerifier.sol

Description Minor: here only checks ‘s.x > 0‘ which is inconsistent with line 85.

```
119 _nextProgress == s.progress + 1 && s.x > 0,
```

CVF-1.55. INFO

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

Recommendation There is much more efficient way to ensure that a big number is greater than one: just check that the bit number is longer than 1 word or that it is exactly one word and this word is greater than 1.

```
132 require(BigNumber.cmp(u, one) == 1, "u must be greater than 1");
```

```
134 require(BigNumber.cmp(u2, one) == 1, "u*u must be greater than 1");
```

CVF-1.56. FIXED

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

Recommendation The concatenation of these values could be pre-hashed, and hash of it could be used here for efficiency.

```
140 s.x,
    s.y.asBytes(nlen),
```

CVF-1.57. INFO

- **Category** Procedural
- **Source** Inflation.sol

Recommendation These variables should be declared as “immutable”.

Client Comment *the VDF difficulty is left mutable for easier governance.*

```
53 uint256 public randomVDFDifficulty;
```

```
63 VDFVerifier public vdfVerifier;
```

CVF-1.58. INFO

- **Category** Suboptimal
- **Source** Inflation.sol

Recommendation A bit mask would be much more efficient.

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

CVF-1.59. FIXED

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

Recommendation Events are usually named via nouns, such as “Claim”, “EntropySeed-Commit”, and “EntropySeedRevelation”.

```
66 event Claimed(address indexed who, uint256 sequence);
```

```
71 event EntropyVDFSeedCommitted(uint256 seed);
```

```
75 event EntropySeedRevealed(bytes32 seed);
```

CVF-1.60. FIXED

- **Category** Suboptimal
- **Source** Inflation.sol

Recommendation Conversions are redundant as “policy” is already “address”.

```
107 address(uint160(policy)),
```

CVF-1.61. FIXED

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

Recommendation The number of Miller-Rabin iterations should be a named constant

```
153 !vdfVerifier.isProbablePrime(x, 10)
```

CVF-1.62. FIXED

- **Category** Suboptimal
- **Source** IsPrime.sol

Description Here linear search is used to find the number of trailing zero bits in “d”, which is suboptimal.

Recommendation Consider using binary search instead like this: if (d « 128 == 0) { d »= 128; s += 128; } if (d « 192 == 0) { d »= 64; s += 64; } if (d « 224 == 0) { d »= 32; s += 32; } if (d « 240 == 0) { d »= 16; s += 16; } if (d « 248 == 0) { d »= 8; s += 8; } if (d « 252 == 0) { d »= 4; s += 4; } if (d « 254 == 0) { d »= 2; s += 2; } if (d « 255 == 0) { d »= 1; s += 1; }

```
53 while (d & 1 == 0) {
       d = d >> 1;
       s++;
   }
```

CVF-1.63. FIXED

- **Category** Suboptimal
- **Source** IsPrime.sol

Description The expression “blockhash(block.number - 1)” is calculated on every loop iteration.

Recommendation Consider calculating once and reusing.

```
60 abi.encode(blockhash(block.number - 1), i)
```

CVF-1.64. FIXED

- **Category** Suboptimal
- **Source** IsPrime.sol

Description The expressions “_n - 4” and “_n - 1” are calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

```
62 uint256 a = (uint256(hash) % (_n - 4)) + 2;
```

```
64 if (x != 1 && x != (_n - 1)) {
```

```
68         if (x == _n - 1) {
```

CVF-1.65. FIXED

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

Description This function implements an overcomplicated padding logic that wound't be necessary if big integer values would be represented as arrays of words rather than arrays of bytes.

```
50 function from(bytes memory _value) internal pure returns (Instance
      ↪ memory) {
```

CVF-1.66. FIXED

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

Recommendation Bitwise "and" would be cheaper.

```
58 if (length % 0x20 == 0x0) {
```

```
121 require(_size % 0x20 == 0x0, "Size must be multiple of 0x20");
```

```
283 assert(_max.length % 0x20 == 0);
    assert(_min.length % 0x20 == 0);
```

CVF-1.67. FIXED

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

Recommendation This could be optimized as: unit256 paddedLength = (length | 0×1f) + 1;

```
77 uint256 paddedLength = 0x20 * ((length + 0x1F) / 0x20);
```

CVF-1.68. FIXED

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

Description These loops process one byte per iteration which is very inefficient.

Recommendation Consider processing one word per iteration.

```
81  for (uint256 i = 0x0; i < offset; ++i) instance.value[i] = 0x0;
    for (uint256 i = 0x0; i < length; ++i)
        instance.value[offset + i] = _value[i];
```

```
128     for (; i < _size - _instance.value.length; ++i) value[i] = 0x0;
        for (; i < _size; ++i)
130         value[i] = _instance.value[
                i - (_size - _instance.value.length)
            ];
```

```
155 for (uint256 i = 0x0; i < size; ++i)
        instance[i] = _base.value[offset + i];
```

CVF-1.69. FIXED

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

Description This loop linearly searches for the first non-zero byte in a word, which is very inefficient.

Recommendation Consider using binary search like this: function countLeadingZeroBytes (bytes32 word) public pure returns (uint256 n) { n = 0; // Redundant, but though if (word » 128 == 0) { word «= 128; n += 16; } if (word » 192 == 0) { word «= 64; n += 8; } if (word » 224 == 0) { word «= 32; n += 4; } if (word » 240 == 0) { word «= 16; n += 2; } if (word » 248 == 0) { word «= 8; n += 1; } if (word == 0) { n += 1; } // No necessary if the word may never be zero }

```
148 while (offset < _base.value.length && _base.value[offset] == 0x0) {
        ++offset;
150 }
```

CVF-1.70. FIXED

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

Recommendation In case “offset” is zero, the original array could be returned as is without copying.

```
155 for (uint256 i = 0x0; i < size; ++i)
        instance[i] = _base.value[offset + i];
```

CVF-1.71. FIXED

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

Description The expression “_a.value.length” is calculated several times.

Recommendation Consider calculating once and reusing.

```
215 assert(_a.value.length % 0x20 == 0);
```

```
218 if (_a.value.length > _b.value.length) return 0x1;
    if (_b.value.length > _a.value.length) return -0x1;
```

```
226 uint256 length = _a.value.length;
```

CVF-1.72. FIXED

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

Description The expression “_b.value.length” is calculated several times.

Recommendation Consider calculating once and reusing.

```
216 assert(_b.value.length % 0x20 == 0);
```

```
218 if (_a.value.length > _b.value.length) return 0x1;
    if (_b.value.length > _a.value.length) return -0x1;
```

CVF-1.73. FIXED

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

Recommendation This line is redundant, as the next line would basically do the same in case both arguments are empty.

```
259 if (_a.value.length == 0x0 && _b.value.length == 0x0) return from("");
```

CVF-1.74. FIXED

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

Recommendation “iszero (eq(i, 0×0))” is equivalent to “gt (i, 0×0)”.

```
304 } iszero(eq(i, 0x0)) {
```

```
417 } iszero(eq(i, 0x0)) {
```

CVF-1.75. FIXED

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

Description Performing this check on every loop iteration is suboptimal.

Recommendation Consider using two consecutive loops: the former goes though the overlapping part of “min” and “max”, while the latter goes through the extra words available only in “max”.

```
309 switch gt(i, sub(max_len, min_len)) // if(i>(max_length-min_length)). while 'min' words are still available.
```

```
422 switch gt(i, len_diff) // if(i>(max_length-min_length)). while 'min' words are still available.
```

CVF-1.76. FIXED

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

Recommendation This could be optimized as: let min_max := add (min_val, max_val) let min_max_carry = add (min_max, carry) mstore (result_ptr, min_max_carry) carry := or (lt (min_max, min_val), lt (min_max_carry, carry))

```
313 mstore(result_ptr, add(add(max_val, min_val), carry)) //
    ↪ result_word = max_word+min_word+carry
```

```
315 switch gt(max_val, sub(uint_max, add(min_val, carry))) //      this
    ↪ switch block finds whether or not to set the carry bit for the
    ↪  next iteration.
case 0x1 {
    carry := 0x1
}
default {
320     switch and(eq(carry, 0x1), eq(min_val, uint_max))
    case 0x1 {
        carry := 0x1
    }
    default {
        carry := 0x0
    }
}
```

CVF-1.77. FIXED

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

Recommendation “sub (uint_max, x)” is equivalent to “not (x)”.

```
315 switch gt(max_val, sub(uint_max, add(min_val, carry))) //      this
    ↪ switch block finds whether or not to set the carry bit for the
    ↪  next iteration.
```

CVF-1.78. FIXED

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

Recommendation This could be optimized as: carry := lt (add (carry, min_val), carry)

```
320 switch and(eq(carry, 0x1), eq(min_val, uint_max))
    case 0x1 {
        carry := 0x1
    }
    default {
        carry := 0x0
    }
```

CVF-1.79. FIXED

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

Recommendation This could be optimized as: let max_carry := add (max_val, carry) mstore (result_ptr, max_carry) carry := lt (max_carry, carry)

```
333 mstore(result_ptr, add(max_val, carry)) //       result_word =
        ↪ max_word+carry
```

```
336 case 0x1 {
        carry := 0x1
    }
    default {
340     carry := 0x0
    }
```

CVF-1.80. FIXED

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

Recommendation This could be optimized as: mstore (result_ptr, carry) carry := shl (0×5, carry) max_len := add (max_len, carry) result := add (result, sub (0×20, carry)) mstore (result, max_len) and the free memory pointer should be updarted right after calculating the initial “result_ptr” value like this: mstore (0×40, add (result_pre, 0×20))

```
347 switch iszero(carry)
    case 0x1 {
        result_start := add(result_start, 0x20)
350 } // if carry is 0x0, increment result_start, ie. length word for
       ↪ result is now one word position ahead.
    default {
        mstore(result_ptr, 0x1)
    } // else if carry is 0x1, store 0x1; overflow has occured, so
       ↪ length word remains in the same position.
```

```
355 result := result_start // point 'result' bytes value to the correct
       ↪ address in memory
    mstore(result, add(max_len, mul(0x20, carry))) // store length of
       ↪ result. we are finished with the byte array.
```

```
358 mstore(0x40, add(result, add(mload(result), 0x20))) // Update
       ↪ freemem pointer to point to new end of memory.
```

CVF-1.81. FIXED

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

Recommendation Left shift would be more efficient than multiplication.

```
356 mstore(result, add(max_len, mul(0x20, carry))) // store length of
       ↪ result. we are finished with the byte array.
```

CVF-1.82. FIXED

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

Recommendation This could be optimized as: let max_min := sub (max_val, min_val) let max_min_carry := sub (max_min, carry) mstore (result_ptr, max_min_carry) carry := or (gt (max_min, max_val), gt (max_min_carry, max_min))

```
426 mstore(result_ptr, sub(sub(max_val, min_val), carry)) //
       ↪ result_word = (max_word-min_word)-carry
```

```
428 switch or(
        lt(max_val, add(min_val, carry)),
430     and(eq(min_val, uint_max), eq(carry, 0x1))
    ) //      this switch block finds whether or not to set the carry
       ↪ bit for the next iteration.
    case 0x1 {
        carry := 0x1
    }
    default {
        carry := 0x0
    }
```

CVF-1.83. FIXED

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

Recommendation This could be optimized as: let max_carry := sub (max_val, carry) mstore (result_ptr, max_carry) carry := gt (max_carry, max_val)

```
444 mstore(result_ptr, sub(max_val, carry)) //      result_word =
       ↪ max_word-carry
```

```
446 switch and(iszero(max_val), eq(carry, 0x1)) //      this switch
       ↪ block finds whether or not to set the carry bit for the next
       ↪ iteration.
    case 0x1 {
        carry := 0x1
    }
450 default {
        carry := 0x0
    }
```

CVF-1.84. FIXED

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

Recommendation It would be more efficient to remember the offset of the most significant non-zero result word when this word is calculating in the loop above, rather than to find it in a separate loop.

```
458 //the following code removes any leading words containing all zeroes
    ↪  in the result.
```

CVF-1.85. FIXED

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

Recommendation As the “diffSquared” value here may never be higher than the “res” value, it is safe to use the “innerDiff” function here instead of “absdiff”.

```
497 res = absdiff(res, diffSquared); // res = add_and_square -
    ↪ diffSquared
```

CVF-1.86. FIXED

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

Recommendation ”mload(_modulus)” here is equivalent to “modIndex”.

```
534 mstore(0x40, add(_modulus, add(mload(_modulus), 0x20))) //update
    ↪ freemem pointer to be modulus index + length
```

CVF-1.87. FIXED

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

Description This makes the “_value” argument redundant.

Recommendation Consider removing it.

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

CVF-1.88. FIXED

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

Description Each word exception for the last one is loaded from the memory twice.

Recommendation Consider refactoring the code to load each word only once and carry bits from one work to another via a local variable.

692
```
wordShifted := mload(resultPtr) //get next word
```

698
```
    precedingWord := mload(sub(resultPtr, 0x20))
```

CVF-1.89. FIXED

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

Description Performing this check on every loop iteration is suboptimal.

Recommendation Consider performing the main loop down to i == 1, and then handling the i == 0 iteration separately after the loop.

693
```
switch iszero(i) //if i==0x0:
```

CVF-1.90. FIXED

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

Recommendation At most one word may become zero during the shift, so this loop could be replaced with a single check.

710
```
//the following code removes any leading words containing all zeroes
   ↪  in the result.
```

CVF-1.91. FIXED

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

Recommendation Events are usually named via nouns, such as “RootHashChallengeIndexRequest”.

```
92  event RootHashChallengeIndexRequestAdded(
101 event ChallengeResponseVerified(
113 event RootHashProposed(
122 event RootHashRejected(
129 event RootHashAccepted(
```

CVF-1.92. FIXED

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

Recommendation These parameters are redundant as their values could be derived from the previous events.

```
93  address indexed proposer,
102 address indexed proposer,
104 address challenger,
123 address indexed proposer,
130 address indexed proposer,
132 uint256 totalSum,
    uint256 amountOfAccounts
139 address indexed proposer,
```

CVF-1.93. FIXED

- **Category** Readability
- **Source** InflationRootHashProposal.sol

Recommendation This parameter should be indexed.

```
96 uint256 index
```

CVF-1.94. FIXED

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

Recommendation This event should have an indexed “index”‘ parameter.

```
138 event ChallengeMissingAccountSuccess(
```

CVF-1.95. FIXED

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

Description This modifier contains much more code than modifiers usually contain. Note, that modifier code is copied for every function that uses the modifier.

Recommendation Consider moving most of the code from this modifier into a function.

```
153 modifier challengeConstraintsAreValid(
```

CVF-1.96. FIXED

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

Description This check in some cases thouls use “>” instead of “>=”.

Recommendation Consider removing it from the modifier and putting into particular fuinctions.

174
```
proposal.amountOfAccounts >= _index,
```

CVF-1.97. FIXED

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

Description The expression “proposal.challenges[_challenger]” is calculated several times.

Recommendation Consider calculating once and reusing.

177
```
uint256 requestsByChallenger = proposal
    .challenges[_challenger]
```

194
```
if (!proposal.challenges[_challenger].initialized) {
```

201
```
        getTime() < proposal.challenges[_challenger].challengeEnds,
```

CVF-1.98. FIXED

- **Category** Readability
- **Source** InflationRootHashProposal.sol

Description It is unclear that the logarithm base here is 2.

Recommendation Consider mentioning this.

182
```
condition  -- x < 2 * log( N ) + 2
              2 ^ x < 2 ^ (2 * log( N ) + 2)
              2 ^ (x - 2) < (2 ^ log( N )) ^ 2
```

CVF-1.99. FIXED

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

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

```
208 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.100. FIXED

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

Recommendation This function should emit some event.

```
215 function configure(uint256 _blockNumber) external {
```

CVF-1.101. FIXED

- **Category** Readability
- **Source** InflationRootHashProposal.sol

Recommendation “co” should be removed.

```
220 /** @notice Allows to propose new root hash co
```

CVF-1.102. FIXED

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

Description The “rootHashProposals[_proposer]” expression is calculated twice: once here and another time inside the “challengeConstraintsAreValid” modifier.

Recommendation Consider refactoring the code to avoid doing the same thing several times.

```
277 RootHashProposal storage proposal = rootHashProposals[_proposer];
```

```
335 RootHashProposal storage proposal = rootHashProposals[_proposer];
```

CVF-1.103. FIXED

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

Description A very similar check was already preformed inside the “challengeConstraintsAreValid” modifier, and this check supersedes that check.

Recommendation Consider refactoring the code to avoid duplicate checks.

```
280 _index < proposal.amountOfAccounts,
```

CVF-1.104. FIXED

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

Recommendation This check is redundant, as it is superseded by the challenge status check below.

```
285 proposal.challengeResponses[_index].account == address(0),
```

CVF-1.105. FIXED

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

Recommendation The “1 days” value should be a named constant.

```
293 challenge.challengeEnds = getTime() + 1 days;
```

CVF-1.106. FIXED

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

Description The expression “proposal.challenges[msg.sender]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
343 proposal.challenges[msg.sender].initialized,
```

```
349     proposal.challenges[msg.sender].challengeStatus[_index - 1] ==
```

```
360     proposal.challenges[msg.sender].challengeStatus[_index] ==
```

CVF-1.107. FIXED

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

Recommendation This argument is redundant, as it could be derived from the “msg.sender” value.

```
390 bytes32 _rootHash,
```

CVF-1.108. FIXED

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

Description The expression “proposal.challengeResponses[_index]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
420 proposal.challengeResponses[_index].account = _account;
    proposal.challengeResponses[_index].balance = _claimedBalance;
    proposal.challengeResponses[_index].sum = _sum;
```

```
443         proposal.challengeResponses[_index].sum == 0,
```

```
456             proposal.challengeResponses[_index].sum,
```

```
461             proposal.challengeResponses[_index].account,
```

```
472         proposal.challengeResponses[_index].sum +
                proposal.challengeResponses[_index].balance ==
```

```
478         proposal.challengeResponses[_index].account <
```

CVF-1.109. FIXED

- **Category** Readability
- **Source** InflationRootHashProposal.sol

Description

Recommendation Should be: else if (proposal.challengeResponses[_index - 1].account != address(0)) {

```
449 if (
450     _index != 0 &&
    proposal.challengeResponses[_index - 1].account != address(0)
) {
```

CVF-1.110. FIXED

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

Description The expression “proposal.challengeResponses[_index - 1]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
451 proposal.challengeResponses[_index - 1].account != address(0)
```

```
454     proposal.challengeResponses[_index - 1].sum +
            proposal.challengeResponses[_index - 1].balance ==
```

```
460     proposal.challengeResponses[_index - 1].account <
```

CVF-1.111. FIXED

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

Recommendation Here the “proposal.challengeResponses[_index].sum” value, that was just written into the storage, is read from the storage again. Just use “_sum” instead.

```
456     proposal.challengeResponses[_index].sum,
```

```
472 proposal.challengeResponses[_index].sum +
```

CVF-1.112. FIXED

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

Recommendation Here the “proposal.challengeResponses[_index].account” value, that was just written into the storage, is read from the storage again. Just use “_account” instead.

```
461     proposal.challengeResponses[_index].account,
```

```
478 proposal.challengeResponses[_index].account <
```

CVF-1.113. FIXED

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

Description The expression “proposal.challengeResponses[_index + 1]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
469 proposal.challengeResponses[_index + 1].account != address(0)
```

```
474         proposal.challengeResponses[_index + 1].sum,
```

```
479         proposal.challengeResponses[_index + 1].account,
```

CVF-1.114. FIXED

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

Recommendation This list could be simplified using the “-=” operator.

```
495 proposal.amountPendingChallenges = proposal.amountPendingChallenges
    ↪ - 1;
```

CVF-1.115. FIXED

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

Recommendation The “1 hours” value should be a named constant.

```
496 challenge.challengeEnds = challenge.challengeEnds + 1 hours;
```

```
699 challenge.challengeEnds = challenge.challengeEnds + 1 hours;
```

CVF-1.116. FIXED

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

Recommendation This line could be simplified using the “+=” operator.

```
496 challenge.challengeEnds = challenge.challengeEnds + 1 hours;
```

CVF-1.117. FIXED

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

Recommendation It would be more efficient to just forbid submitting several identical proposals.

```
581 (proposal.status == RootHashStatus.Rejected &&
        _rootHash == acceptedRootHash &&
        proposal.totalSum == acceptedTotalSum &&
        proposal.amountOfAccounts == acceptedAmountOfAccounts),
```

CVF-1.118. FIXED

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

Description The expression “proposal.challenges[_who]” is calculated twice.

Recommendation Consider calculating one and reusing.

```
590     proposal.challenges[_who].initialized &&
```

```
594 uint256 amount = proposal.challenges[_who].amountOfRequests *
```

CVF-1.119. FIXED

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

Recommendation The conversions are redundant, as “policy” is already “address”.

```
626 address(uint160(policy)),
```

CVF-1.120. FIXED

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

Recommendation The conversion is redundant, as “_root” is already “bytes32”.

```
687 return computedHash == bytes32(_root);
```

CVF-1.121. FIXED

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

Recommendation These lines could be simplified using the “+=” operator.

```
696 proposal.totalChallenges = proposal.totalChallenges + 1;
proposal.amountPendingChallenges = proposal.amountPendingChallenges
    ↪ + 1;
challenge.amountOfRequests = challenge.amountOfRequests + 1;
```

CVF-1.122. FIXED

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

Description The “challenge.challengeEnds” value is read from the storage twice.

Recommendation Consider reading once and reusing.

```
701 if (proposal.lastLiveChallenge < challenge.challengeEnds) {
        proposal.lastLiveChallenge = challenge.challengeEnds;
```

CVF-1.123. FIXED

- **Category** Readability
- **Source** InflationRootHashProposal.sol

Recommendation The name should be “_proposer’ rather than “_proposal”.

```
710 address _proposal,
```

CVF-1.124. FIXED

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

Recommendation This line could be optimized via the “+=” operator.

```
714 rootHashProposals[_proposal].stakedAmount =
        rootHashProposals[_proposal].stakedAmount +
        _fee;
```

CVF-1.125. FIXED

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

Description The expression “rootHashProposals[_proposal]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
714 rootHashProposals[_proposal].stakedAmount =
        rootHashProposals[_proposal].stakedAmount +
```

CVF-1.126. FIXED

- **Category** Procedural
- **Source** InflationRootHashProposal.sol

Description The contract doesn’t expect that value to change, as after changing the token address, fees collected in old tokens will be effectively lost.

Recommendation Consider either implementing some way to rescue tokens other that the token returned by this call, or to obtain the token address from the policy once in the constructor and storing in an immutable variable to ensure that it will not change.

Client Comment *The contract is instead sets an immutable address in the constructor for the token address.*

```
722 return IERC20(policyFor(ID_ERC20TOKEN));
```

CVF-1.127. FIXED

- **Category** Documentation
- **Source** ECOx.sol

Description The meaning of this value is unclear.

Recommendation Consider explaining in a documentation comment.

```
27 uint8 public constant PRECISION = 100;
```

CVF-1.128. FIXED

- **Category** Procedural
- **Source** ECOx.sol

Recommendation This variable should be declared as immutable.

```
29 uint256 public initialSupply;
```

CVF-1.129. FIXED

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

Description There is not range check for the “_initialSupply” value.

Recommendation Consider adding appropriate checks.

```
31 constructor(address _policy, uint256 _initialSupply)
```

CVF-1.130. FIXED

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

Recommendation This assignment wouldn’t be necessary if the “initialSupply” variable would be declared as “immutable“.

```
41 initialSupply = ECOx(_self).initialSupply();
```

CVF-1.131. FIXED

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

Description This check is redundant for a properly initialized contract.

Recommendation Consider removing it.

```
66 require(initialSupply > 0, "initial supply not set");
```

CVF-1.132. INFO

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

Recommendation It would probably be more precise to estimate e^x in the following way: https://hackmd.io/@abdk/HyFS9a66K

Client Comment *The error of the binary fraction method is that it cannot differentiate past the number of binary digits corresponding to precomputed coefficients. The error of the taylor series is based on the size of the final term, which means that for conversion fractions of 10% and lower, the current implementation already beats out a 64 binary coefficient estimation and can beat out a 128 digit one with only the addition of 5 more terms. The max decimal digits of our fraction is 27, so at least 90 binary fraction coefficients would need to be calculated for single wei precision. Gas efficiency wise, our calculation takes two multiplications, one addition, two variable storages, and one right shift per term. The binary fraction expansion takes one multiplication, one comparison, one right shift, and one variable storage per term (worst case). On average, only every other term is calculated, which means that the Taylor series method is slightly worse than twice as gas inefficient per term. However, in requiring less than one third the number of terms to hit maximum accuracy within the range in question, our calculation is still more gas efficient. For all these reasons, a Taylor expansion is more suited for our implementation.*

```
73 * @dev this function can be auto-generated by the script '
      ↪ PrintFunctionGeneralExp.py'.
   * it approximates "e ^ x" via maclaurin summation: "(x^0)/0! + (x^1)
      ↪ /1! + ... + (x^n)/n!".
   * it returns "e ^ (x / 2 ^ precision) * 2 ^ precision", that is, the
      ↪  result is upshifted for accuracy.
   * the global "maxExpArray" maps each "precision" to "((
      ↪ maximumExponent + 1) << (MAX_PRECISION - precision)) - 1".
   * the maximum permitted value for "x" is therefore given by "
      ↪ maxExpArray[precision] >> (MAX_PRECISION - precision)".
```

CVF-1.133. FIXED

- **Category** Procedural
- **Source** EcoBalanceStore.sol

Description The names of some other interfaces in this project are prefixed with “I” while this interface name is not prefixed.

Recommendation Consider using a consistent naming policy.

```
7 interface EcoBalanceStore {
```

CVF-1.134. FIXED

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

Description The semantics of the “uint256” arguments is unclear.

Recommendation Consider giving descriptive names fo these arguments and/or describing their semantics in a documentation comment.

```
22 function balanceAt(address, uint256) external view returns (uint256)
      ↪ ;
```

```
24 function totalSupplyAt(uint256) external view returns (uint256);
```

CVF-1.135. FIXED

- **Category** Readability
- **Source** InflationCheckpoints.sol

Recommendation This number could be rendered as “1e18”.

```
20 1_000_000_000_000_000_000;
```

CVF-1.136. INFO

- **Category** Flaw
- **Source** InflationCheckpoints.sol

Description Visible token balances for this contract may change on their own without emitting any events. This could make it impossible to user this token with some existing DeFi applications, such as Uniswap V3, as many DeFi application do their own accounting for token balances, and this internal accounting will not match actual balances after changing the linear inflation quotient.

Recommendation Consider wrapping this token in another token whose balances are stable but the amount of the inner token per one unit of the outer token vary.

Client Comment *We have a wrapper contract, but it is external to the core currency.*

```
82 function balance(address _owner) public view override returns (
      ↪ uint256) {
```

CVF-1.137. FIXED

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

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are voter addresses, and values are structs with two fields encapsulating the values of the original mappings.

```
20 mapping(address => uint256) public stake;
```

```
24 mapping(address => bool) public yesVote;
```

CVF-1.138. FIXED

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

Recommendation Events are usually named via nouns, such as “VoteCompletion” or “PolicyVote”.

```
55 event VoteCompleted(Result result);
```

```
59 event PolicyVoteCast(address indexed voter, bool vote, uint256
   ↪ amount);
```

CVF-1.139. FIXED

- **Category** Procedural
- **Source** PolicyVotes.sol

Recommendation The “result” parameter should be indexed.

```
55 event VoteCompleted(Result result);
```

CVF-1.140. FIXED

- **Category** Procedural
- **Source** PolicyVotes.sol

Recommendation The “vote” parameter should be indexed.

```
59 event PolicyVoteCast(address indexed voter, bool vote, uint256
   ↪ amount);
```

CVF-1.141. FIXED

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

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

```
66 constructor(address _policy) VotingPower(_policy) {}
```

CVF-1.142. FIXED

- **Category** Readability
- **Source** PolicyVotes.sol

Recommendation These lines could be simplified using “+=” and “-=” operators.

```
95         yesStake = yesStake - _oldStake;
98     totalStake = totalStake - _oldStake;
106    yesStake = yesStake + _amount;
111 totalStake = totalStake + _amount;
```

CVF-1.143. FIXED

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

Description The “yesStake”, “yesVotes[msg.sender]”, “totalStake”, and “stake[msg.sender]” values are potentially updated twice.

Recommendation Consider refactoring the code to update each value at most once.

```
95          yesStake = yesStake - _oldStake;
            yesVote[msg.sender] = false;
```

```
98      totalStake = totalStake - _oldStake;
        stake[msg.sender] = 0;
```

```
106     yesStake = yesStake + _amount;
        yesVote[msg.sender] = true;
```

```
109 stake[msg.sender] = _amount;
```

```
111 totalStake = totalStake + _amount;
```

CVF-1.144. FIXED

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

Description This should be executed only when “yesStake <= _total / 2” is true.

```
152 uint256 _time = getTime();
```

CVF-1.145. INFO

- **Category** Procedural
- **Source** PolicyVotes.sol

Recommendation All this code should probably be executed before enacting the proposal, as the proposal could do some policy changes incompatible with this code.

Client Comment *The proposed change is not possible since you need the permissions being revoked here to execute the policy.*

```
178 emit VoteCompleted(_res);
    Policy(policy).removeSelf(ID_POLICY_VOTES);
```

```
181 getToken().transfer(
        address(uint160(policy)),
        getToken().balanceOf(address(this))
    );
```

CVF-1.146. FIXED

- **Category** Suboptimal
- **Source** TokenEvents.sol

Description This interface is not used.

Recommendation Consider removing it.

Client Comment *this file is deleted*

```
6 interface TokenEvents {
```

CVF-1.147. FIXED

- **Category** Procedural
- **Source** TokenEvents.sol

Recommendation If these functions are supposed to emit some events, these events should be defined in this interface.

Client Comment *this file is deleted*

```
8  function emitSentEvent(
```

```
18 function emitMintedEvent(
```

```
27 function emitBurnedEvent(
```

CVF-1.148. INFO

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

Description This contract allows retrieving historical balances for all past blocks which could be an overkill. If the historical balances are needed only for certain block numbers, it would be more efficient to maintain a snapshot counter which is incremented every time a snapshot of balances is created, and the snapshot counter value should be used instead of the block number when writing checkpoints.

Client Comment *This approach was explored previously to this implementation and it came with its own issues. Notably, there needed to be an update method that makes sure that the most recent snapshots are filled forward with previous snapshots if there had been no changes to the balance which caused unpredictable gas costs on seemingly simple functions.*

```
22 abstract contract VoteCheckpoints is ERC20 {
```

CVF-1.149. FIXED

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

Recommendation Events are usually named via nouns, such as “Delegate” and “DelegateVote”.

```
38 event DelegateChanged(
```

```
47 event DelegateVotesChanged(
```

CVF-1.150. FIXED

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

Description These parameters are redundant as they contain previous value that could be derived from previous events.

```
40 address indexed fromDelegate,
```

```
49 uint256 previousBalance,
```

CVF-1.151. FIXED

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

Description These functions are just aliases for the “balanceOf” and “totalSupply” functions.

Recommendation Consider removing these functions or making them non-public.

```
67 function balance(address _owner) public view virtual returns (
   ↪ uint256) {
```

```
73 function tokenSupply() public view virtual returns (uint256) {
```

CVF-1.152. FIXED

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

Recommendation The function name is confusing, as it returns a single checkpoint, not several of them.

```
111 function checkpoints(address account, uint32 pos)
```

CVF-1.153. FIXED

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

Description Overflow is possible when converting to “uint32”.

Recommendation Consider using safe conversion.

```
130 return uint32(_checkpoints[account].length);
```

CVF-1.154. FIXED

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

Recommendation The function name is confusing, as the function returns a single delegate, not several of them.

```
136 function delegates(address account) public view virtual returns (
    ↪ address) {
```

CVF-1.155. FIXED

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

Description The expression “ckpts.length” is calculated several times.

Recommendation Consider calculating once and reusing.

```
210 if (ckpts.length == 0) return 0;
    if (blockNumber > ckpts[ckpts.length - 1].fromBlock)
        return ckpts[ckpts.length - 1].value;
```

```
214 uint256 high = ckpts.length;
```

CVF-1.156. FIXED

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

Description The expression “ckpts[ckpts.length - 1]” is potentially calculated twice.

Recommendation Consider calculating once and reusing.

```
211 if (blockNumber > ckpts[ckpts.length - 1].fromBlock)
        return ckpts[ckpts.length - 1].value;
```

CVF-1.157. FIXED

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

Recommendation This expression could be optimized as: low + ((high - low) » 1)

```
217 uint256 mid = (low & high) + (low ^ high) / 2;
```

CVF-1.158. FIXED

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

Recommendation It would be simpler and more flexible to just pass the new value, instead of a function and its argument..

```
332 function(uint256, uint256) view returns (uint256) op,
    uint256 delta
```

CVF-1.159. FIXED

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

Description The expression “ckpts[pos - 1]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
336 oldWeight = pos == 0 ? 0 : ckpts[pos - 1].value;
```

```
348 if (pos > 0 && ckpts[pos - 1].fromBlock == block.number) {
        ckpts[pos - 1].value = uint224(newWeight);
```

CVF-1.160. INFO

- **Category** Procedural
- **Source** TrusteeReplacement.propo.sol

Recommendation This contract should be defined in a file named “TrusteeReplacement.sol”.

Client Comment *Proposal contracts are marked with the file postfix as they all follow a set template specification.*

```
12 contract TrusteeReplacement is Policy, Proposal {
```

CVF-1.161. INFO

- **Category** Procedural
- **Source** TrusteeReplacement.propo.sol

Description Template contracts that ought to be modified before deployment are errorrone.

Recommendation Consider using abstract contracts instead that have abstract function for those parts of functionality that ought to be different for particular proposals. inherit particular proposals from these abstract contracts.

```
27 return "Trustee Election Proposal Template";
```

```
33 return
       "Created with a list of trustees and replaces all current
          ↪ trustees with those trustees";
```

```
40 return
       "https://description.of.proposal make this link to a discussion
          ↪ of the new trustee slate";
```

CVF-1.162. FIXED

- **Category** Procedural
- **Source** IEcoBalanceStoreGenerationBalance.sol

Description Some other contracts have function with the same signature, but with a different semantics for the second argument, which here specifies a generation while in other contracts specifies a block number.

Recommendation Consider renaming to “balanceAtGeneration”.

```
5 function balanceAt(address _owner, uint256 _pastGeneration)
```

CVF-1.163. FIXED

- **Category** Suboptimal
- **Source** ERC20.sol

Description While Solidity doesn't support immutable string variables, it is possible to cheaply pack/unpack a short string into a bytes32 value that could be stored in an immutable variable.

Recommendation See the following gist for how this could be done: https://gist.github.com/3sGgpQ8H/567354534170905e047b299286697e19

```
39 string internal _name;
40 string internal _symbol;
```

CVF-1.164. INFO

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

Description This emits an "Approval" event not expected here according to the ERC-20 standard.

Recommendation Consider not emitting unexpected events.

Client Comment *This event allows for complete accounting. Even if other tokens may not have it, the system integrety is improved by its inclusion.*

```
196 _approve(sender, msg.sender, currentAllowance - amount);
```

CVF-1.165. INFO

- **Category** Unclear behavior
- **Source** ERC20.sol

Description Transferring token to zero address is a common way to effectively burn them.

Recommendation Consider not forbidding such transfers.

Client Comment *We want to limit burning tokens to approved functions. Explicitly burning works as a double-check that the user did not mean to instead transfer the tokens.*

```
277 require(recipient != address(0), "ERC20: transfer to the zero
    ↪ address");
```

CVF-1.166. INFO

- **Category** Suboptimal
- **Source** ERC20.sol

Description Many existing contracts expect that a successful token transfer transfers exactly as man y tokens as was requested, and don't do balance checks before and after the transfer to find out the actual amount of tokens sent. Amending the transfer amount could make this ERC-20 implementation incompatible with such contracts.

Recommendation Consider not amending transfer amounts in order to comply with the standard.

Client Comment *We have a rebasing token, so this is not possible. Our balance functions are all overridden and this function covers the mechanics of the conversion to underlying balances. Instead there is a separate event that tracks these values that can be used to not have to directly check the balance.*

```
279 uint256 amount = _beforeTokenTransfer(
```

CVF-1.167. INFO

- **Category** Suboptimal
- **Source** ERC20.sol

Description This check looks redundant. It is anyway possible to mint token to a dead address.

Recommendation Consider removing this check.

Client Comment *We wish to not have dead tokens at dead addresses as the total supply is an important metric for governance*

```
314 require(account != address(0), "ERC20: mint to the zero address");
```

CVF-1.168. INFO

- **Category** Suboptimal
- **Source** ERC20.sol

Description This check looks redundant. What problem it tries to solve?

Recommendation Consider removing it.

Client Comment *This stops users from thinking they can send tokens to the zero address.*

```
385 require(spender != address(0), "ERC20: approve to the zero address")
        ↪ ;
```

CVF-1.169. FIXED

- **Category** Bad naming
- **Source** ERC20.sol

Description The semantics of these arguments is unclear without names.

Recommendation Consider giving descriptive names to all the arguments and/or describing their semantics in the documentation comment.

```
406 address,
    address,
```

CVF-1.170. INFO

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

Description These variables should be declared as immutable.

Client Comment *Making this immutable requires redeploying this contract if a change is made to any of these files. To allow for easier governance, this is left mutable.*

```
34 address public policyProposalImpl;
```

```
42 address public simplePolicyImpl;
```

CVF-1.171. FIXED

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

Recommendation The type of this variable should be “PolicedUtils”.

```
34 address public policyProposalImpl;
```

CVF-1.172. FIXED

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

Recommendation The type of this variable should be “SimplePolicySetter”.

```
42 address public simplePolicyImpl;
```

CVF-1.173. FIXED

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

Recommendation Events are usually named via nouns, such as “PolicyDecisionStart”.

```
52 event PolicyDecisionStarted(address contractAddress);
```

CVF-1.174. FIXED

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

Recommendation The types of these arguments should be “PolicedUtils”, and “Simple-PolicySetter” respectively.

```
56 address _policyproposal,
   address _simplepolicy,
```

CVF-1.175. INFO

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

Recommendation These assignments wouldn't be necessary if the "policyProposalImpl" and the "simpePolicyImpl" variables would be declared as immutable.

Client Comment *This is left mutable for easier governance.*

```
68 policyProposalImpl = TimedPolicies(_self).policyProposalImpl();
   simplePolicyImpl = TimedPolicies(_self).simplePolicyImpl();
```

CVF-1.176. FIXED

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

Description The 'getTime()' is called twice here.

Recommendation Consider caching or inlining.

```
82     getTime() > nextGenerationStart,
```

```
86 nextGenerationStart = getTime() + GENERATION_DURATION;
```

CVF-1.177. FIXED

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

Description The "notificationHashes.length" expression is calculated on every loop iteration.

Recommendation Consider calculating once and reusing.

```
89 for (uint256 i = 0; i < notificationHashes.length; ++i) {
```

CVF-1.178. FIXED

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

Description This function is redundant.

Recommendation Just rename “internalGenration” to “generation” and make it “public override” property.

```
100 function generation() external view override returns (uint256) {
```

CVF-1.179. FIXED

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

Description Deploying a separate contract just to use it once is suboptimal.

Recommendation Consider implementing an ability pass additional data to the called contract through the “internalCommand” function, so a single proposal instance could be reusing.

```
116 SimplePolicySetter sps = SimplePolicySetter(
```

CVF-1.180. INFO

- **Category** Procedural
- **Source** SingleTrusteeReplacement.propo.sol

Recommendation This contract should be defined in a file named “SingleTrusteeReplacement.sol”.

Client Comment *Proposal contracts are marked with the file postfix as they all follow a set template specification.*

```
12 contract SingleTrusteeReplacement is Policy, Proposal {
```

CVF-1.181. FIXED

- **Category** Procedural
- **Source** SingleTrusteeReplacement.propo.sol

Description These variables should be declared as immutable.

```
14 address public oldTrustee;
```

```
17 address public newTrustee;
```

CVF-1.182. INFO

- **Category** Suboptimal
- **Source** SingleTrusteeReplacement.propo.sol

Description There is no check that the old and the new trustees are different.

Recommendation Consider adding such check.

Client Comment *There are several incorrect types of inputs. As this is a proposal, it will simply be rejected if the inputs are non-sensical.*

```
24 constructor(address _oldTrustee, address _newTrustee) {
```

CVF-1.183. INFO

- **Category** Suboptimal
- **Source** SingleTrusteeReplacement.propo.sol

Description Template code has to be modified before deployment which is error-prone.

Recommendation Consider using abstract contract instead of template contract, with abstract functions for the parts that ought to be implemented in particular proposals.

```
32 return "Trustee Replacement Proposal Template";
```

```
38 return "Replaces as single trustee with another";
```

```
45     "https://description.of.proposal make this link to a discussion
         ↪ of the no confidence vote";
```

CVF-1.184. FIXED

- **Category** Documentation
- **Source** SingleTrusteeReplacement.propo.sol

Recommendation as → a

```
38 return "Replaces as single trustee with another";
```

CVF-1.185. FIXED

- **Category** Unclear behavior
- **Source** SingleTrusteeReplacement.propo.sol

Description This should be a compile-time constant.

```
55 bytes32 _trustedNodesId = keccak256(abi.encodePacked("TrustedNodes")
   ↪ );
```

CVF-1.186. FIXED

- **Category** Suboptimal
- **Source** SingleTrusteeReplacement.propo.sol

Recommendation This assignment wouldn't be necessary in case the "oldTrustee" and the "newTrustee" variables would be immutable.

```
58 address _oldTrustee = SingleTrusteeReplacement(_self).oldTrustee();
   address _newTrustee = SingleTrusteeReplacement(_self).newTrustee();
```

CVF-1.187. FIXED

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

Recommendation Events are usually named via nouns, such as "InflationRootHashProposalStart".

```
17 event InflationRootHashProposalStarted(
```

CVF-1.188. FIXED

- **Category** Readability
- **Source** CurrencyTimer.sol

Recommendation The parameter should be indexed.

```
18 address inflationRootHashProposalContract,
```

CVF-1.189. FIXED

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

Description This variable should be declared as immutable.

```
28 InflationRootHashProposal public inflationRootHashProposalImpl;
```

CVF-1.190. FIXED

- **Category** Suboptimal
- **Source** CurrencyTimer.sol

Recommendation This assignment wouldn't be necessary in case the "inflationRootHashProposalImpl" variable was immutable.

Client Comment *Making this immutable requires redeploying this contract if a change is made to inflationRootHashProposal.sol. To allow for easier governance, this is left mutable.*

```
39 inflationRootHashProposalImpl = EcoBalanceStore(_self)
40     .inflationRootHashProposalImpl();
```

CVF-1.191. FIXED

- **Category** Suboptimal
- **Source** ECO.sol

Recommendation This assignment is redundant, as the assigned value is overwritten in the next line.

```
87 uint256 _inflationMultiplier = INITIAL_INFLATION_MULTIPLIER;
```

CVF-1.192. FIXED

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

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

22
```
mapping(uint256 => address[]) public trustedNodes;
```

25
```
mapping(uint256 => mapping(address => uint256)) public trusteeNumber
    ↪ ;
```

CVF-1.193. FIXED

- **Category** Documentation
- **Source** TrustedNodes.sol

Description Should probably be “Address of trusted node” instead of “Index of trusted node”.

Recommendation Consider explaining more clearly the meaning of the keys and values of this mapping.

24
```
/** @dev Index of trusted node to position in trustedNodes per
    ↪ cohort */
```

CVF-1.194. FIXED

- **Category** Documentation
- **Source** TrustedNodes.sol

Description A per-trustee value inside this mapping is not only incremented on every trustee vote, but also is reset to zero when the trustee redeems vote rewards.

Recommendation Consider mentioning this in the comment.

27
```
/** Increments each time the trustee votes */
```

CVF-1.195. FIXED

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

Recommendation Events are usually named via nouns, such as “TrusteeNode”, “TrustyNodeRemoval”, “VotingRewardRedemption”.

```
34 event TrustedNodeAdded(address indexed node);
```

```
38 event TrustedNodeRemoved(address indexed node);
```

```
41 event VotingRewardRedeemed(address indexed trustee, uint256 amount);
```

CVF-1.196. FIXED

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

Description The name is confusing.

Recommendation Consider renaming to “initialTrustedNodes”.

```
47 address[] memory _initial,
```

CVF-1.197. INFO

- **Category** Flaw
- **Source** TrustedNodes.sol

Description There are no range checks for this argument.

Recommendation Consider adding appropriate checks.

```
48 uint256 _voteReward
```

CVF-1.198. FIXED

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

Description These calls add trusted nodes without emitting “TrustedNodeAdded” events. This makes it harder to to track the current set of trusted nodes.

Recommendation Consider emitting appropriate events.

```
50  _trust(address(0));
```

```
53      _trust(_initial[i]);
```

```
149 _trust(address(0));
```

```
152     _trust(_newCohort[i]);
```

CVF-1.199. FIXED

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

Description Adding the zero address as a trusted node looks like a hack. This is probably done to make it possible to know whether a node is trusted or not.by comparing the “trustedNumber[cohort][node]” with zero.

Recommendation Consider implementing a more elegant approach, e.g. compare trustedNodes [trustedNumber[cohort][node]] with node.

```
50  _trust(address(0));
```

```
149 _trust(address(0));
```

CVF-1.200. FIXED

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

Description The expression “trustedNumber[cohort]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
77 require(trusteeNumber[cohort][_node] > 0, "Node already not trusted"
   ↪ );
```

```
79 uint256 oldIndex = trusteeNumber[cohort][_node];
```

```
82 delete trusteeNumber[cohort][_node];
```

```
88     trusteeNumber[cohort][lastNode] = oldIndex;
```

CVF-1.201. FIXED

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

Description The expression “trustedNumber[cohort][_node]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
77 require(trusteeNumber[cohort][_node] > 0, "Node already not trusted"
   ↪ );
```

```
79 uint256 oldIndex = trusteeNumber[cohort][_node];
```

CVF-1.202. FIXED

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

Description The expression “trustedNodes[cohort]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
80  uint256 lastIndex = trustedNodes[cohort].length - 1;
```

```
85      address lastNode = trustedNodes[cohort][lastIndex];
```

```
87      trustedNodes[cohort][oldIndex] = lastNode;
```

```
91  delete trustedNodes[cohort][lastIndex];
    trustedNodes[cohort].pop();
```

CVF-1.203. FIXED

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

Description The “delete” statement is redundant, as the “pop” function implicitly performs “delete” on removed element.

Recommendation See documentation for details: https://docs.soliditylang.org/en/develop/types.html#array-members

```
91  delete trustedNodes[cohort][lastIndex];
    trustedNodes[cohort].pop();
```

CVF-1.204. FIXED

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

Description The expression “trustedNumber[cohort]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
133 require(trusteeNumber[cohort][_node] == 0, "Node is already trusted"
    ↪ );
```

```
135 trusteeNumber[cohort][_node] = trustedNodes[cohort].length;
```

CVF-1.205. FIXED

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

Description The expression “trustedNodes[cohort]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
135 trusteeNumber[cohort][_node] = trustedNodes[cohort].length;
    trustedNodes[cohort].push(_node);
```

CVF-1.206. FIXED

- **Category** Procedural
- **Source** VotingPower.sol

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

```
15 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.207. FIXED

- **Category** Suboptimal
- **Source** SimplePolicySetter.sol

Description It is possible to clone from the outside of this contract. Also, this function could be called on a clone effectively cloning a proxy rather than the original implementation.

Recommendation Consider removing this function.

Client Comment *This contract is removed.*

```
51 function clone(bytes32 _key, address _value) external returns (
   ↪ address) {
```

CVF-1.208. FIXED

- **Category** Procedural
- **Source** Proposal.sol

Recommendation This abstract contract could be turned into an interface.

```
8 abstract contract Proposal {
```

CVF-1.209. INFO

- **Category** Suboptimal
- **Source** Proposal.sol

Recommendation It would be cheaper to just emit these values in an event when a proposal is created or when any of these values change. For off-chain application there is not big difference between calling a read-only function on a contract or querying for an event emitted from this contract.

Client Comment *We prefer this pattern here as the proposals will likely be considered in un-deployed form as well.*

```
13 function name() external view virtual returns (string memory);
```

```
17 function description() external view virtual returns (string memory)
   ↪ ;
```

```
22 function url() external view virtual returns (string memory);
```

CVF-1.210. FIXED

- **Category** Suboptimal
- **Source** Lockup.sol

Description These two statements import the same file.

Recommendation Consider removing one of them.

```
5 import "../governance/IGeneration.sol";
```

```
8 import "./IGeneration.sol";
```

CVF-1.211. FIXED

- **Category** Readability
- **Source** Lockup.sol

Recommendation The “to” parameter should be indexed.

```
32 event Sale(address to, uint256 amount);
```

```
42 event Withdrawal(address to, uint256 amount);
```

CVF-1.212. FIXED

- **Category** Procedural
- **Source** Lockup.sol

Recommendation The name “Deposit” would be more conventional.

```
32 event Sale(address to, uint256 amount);
```

CVF-1.213. FIXED

- **Category** Readability
- **Source** Lockup.sol

Recommendation This value could be rendered as 1e9.

```
48 uint256 public constant BILLION = 1_000_000_000;
```

CVF-1.214. FIXED

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

Description The constant name is helpless, as it doesn't add anything to the constant value.

Recommendation Consider renaming to "INTEREST_DENOMINATOR".

```
48 uint256 public constant BILLION = 1_000_000_000;
```

CVF-1.215. FIXED

- **Category** Documentation
- **Source** Lockup.sol

Description The number format of this variable is unclear.

Recommendation Consider explaining in a documentation comment.

```
50 uint256 public interest;
```

CVF-1.216. FIXED

- **Category** Suboptimal
- **Source** Lockup.sol

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are depositor addresses and values are structs with two fields encapsulating the values of the original mappings.

```
54 mapping(address => uint256) public depositBalances;
   mapping(address => uint256) public depositLockupEnds;
```

CVF-1.217. FIXED

- **Category** Procedural
- **Source** Lockup.sol

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

```
57 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.218. FIXED

- **Category** Suboptimal
- **Source** Lockup.sol

Description This function is redundant, as cloning could be done from the outside of the contract. Also, when called on a clone, this function will clone a proxy rather than the original implementation.

Recommendation Consider removing this function.

Client Comment *We prefer cloning from a template instead of creating a clone externally. This function needs to exist to overload the underlying one in PolicedUtils.sol as there are additional params that need to be passed.*

```
71 function clone(uint256 _duration, uint256 _interest)
```

CVF-1.219. FIXED

- **Category** Procedural
- **Source** Lockup.sol

Description There are no range checks of the arguments.

Recommendation Consider adding appropriate checks.

```
71 function clone(uint256 _duration, uint256 _interest)
```

```
82     uint256 _duration,
       uint256 _interest
```

CVF-1.220. FIXED

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

Recommendation The return type should be “Lockup”.

```
73 returns (address)
```

CVF-1.221. FIXED

- **Category** Suboptimal
- **Source** Lockup.sol

Description The “totalDeposit” value is read from the storage twice.

Recommendation Consider reading once and reusing.

```
93 totalDeposit +
   (totalDeposit * interest) /
```

CVF-1.222. INFO

- **Category** Suboptimal
- **Source** Lockup.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary operation overflows.

Recommendation Consider using a non-overflowing “muldiv” function such as described here: https://xn–2-umb.com/21/muldiv/index.html.

```
94     (totalDeposit * interest) /
       BILLION -
```

```
112 uint256 _delta = (_amount * interest) / BILLION;
```

CVF-1.223. FIXED

- **Category** Readability
- **Source** Lockup.sol

Recommendation These lines could be simplified using “+=” and “-=” operators.

```
111 totalDeposit = totalDeposit - _amount;
```

```
115     _amount = _amount - _delta;
```

```
117     _amount = _amount + _delta;
```

```
139 totalDeposit = totalDeposit + _amount;
140 depositBalances[_who] = depositBalances[_who] + _amount;
```

CVF-1.224. FIXED

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

Recommendation The name should be “Stage” as a single enum value is a single stage.

```
19 enum Stages {
```

CVF-1.225. FIXED

- **Category** Documentation
- **Source** CurrencyGovernance.sol

Description The semantics of keys for these mappings is unclear.

Recommendation Consider documenting.

```
42 mapping(address => GovernanceProposal) public proposals;
   mapping(address => bytes32) public commitments;
   mapping(address => uint256) public score;
```

```
46 mapping(address => bool) internal voteCheck;
```

CVF-1.226. FIXED

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

Recommendation Events are usually named via nouns, such as “Revelation” and “Winner”.

```
59 event VoteRevealed(address indexed voter, address[] votes);
```

```
64 event VoteResults(address winner);
```

CVF-1.227. FIXED

- **Category** Procedural
- **Source** CurrencyGovernance.sol

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

```
83 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.228. INFO

- **Category** Procedural
- **Source** CurrencyGovernance.sol

Description This function doesn't check that a proposal from the same address already exists.

Recommendation Consider adding such check.

Client Comment *This check isn't done as trustees are allowed to modify their proposal at any time up until voting. Skipping this check saves them from having to call unpropose and then propose.*

```
95 function propose(
```

CVF-1.229. FIXED

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

Description These functions should emit some events.

```
95 function propose(
```

```
111 function unpropose() external atStage(Stages.Propose) {
```

```
115 function commit(bytes32 _commitment)
```

CVF-1.230. INFO

- **Category** Documentation
- **Source** CurrencyGovernance.sol

Description This function presumes that trustees are trusted with submitting only actual votes, and can't steal a vote from each other.

Recommendation Consider making this explicit in the documentation.

Client Comment *This function assumes that the commitment is of a vote. If a trustee commits something that isn't a vote, they have forgone their right to vote. Because the msg.sender is in the hash commit and also this function looks up your commit by msg.sender, I don't see what you mean by "steal a vote". The timescale of voting makes it unlikely for the hash to be cracked, but in doing so you gain the information of what their vote is, not any ability to manipulate the reveal function.*

```
123 function reveal(bytes32 _seed, address[] calldata _votes)
```

CVF-1.231. FIXED

- **Category** Procedural
- **Source** CurrencyGovernance.sol

Recommendation This check should be performed at the very beginning of the function.

```
131 require(_votes.length > 0, "Cannot vote empty");
```

CVF-1.232. FIXED

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

Recommendation These loops should be merged into a single loop while still forbidding double voting.

```
141 for (uint256 i = 0; i < _votes.length; ++i) {
```

```
154 for (uint256 i = 0; i < _votes.length; ++i) {
```

CVF-1.233. FIXED

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

Description Using an in-storage mapping here for preventing double voting for the same address is suboptimal.

Recommendation Consider just requiring the addresses to in ascending order.

Client Comment *Borda count is an ordered list voting schema so the order matters. Instead an in-memory array can be used as it does not need to persist past this loop, but needs to be sorted which comes with a scaling tradeoff.*

```
143 require(!voteCheck[v], "Repeated vote");
```

```
146 voteCheck[v] = true;
```

```
155 voteCheck[_votes[i]] = false;
```

CVF-1.234. FIXED

- **Category** Flaw
- **Source** CurrencyGovernance.sol

Description This function doesn't distinguish between real winner and the situation when no real proposal won and "leader" is zero address.

Recommendation Consider distinguishing these situations, emitting different events in them, and transitioning into different states.

Client Comment *The zero address being the leader and the winner is a valid situation that will occur. It represents the default proposal of no changes winning.*

```
166 winner = leader;
```

CVF-1.235. FIXED

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

Recommendation The type of this field should be "Proposal".

```
38 address proposal;
```

CVF-1.236. FIXED

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

Recommendation The name should be “totalStake” in camelCase.

```
41 uint256 totalstake;
```

CVF-1.237. FIXED

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

Recommendation The name should be “totalProposals” in camelCase.

```
54 uint256 public totalproposals;
```

CVF-1.238. FIXED

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

Recommendation The type of this variable should be “Proposal []”.

```
58 address[] public allProposals;
```

CVF-1.239. FIXED

- **Category** Readability
- **Source** PolicyProposals.sol

Recommendation This value could be rendered as “1000e18” or “1e21”.

```
70 uint256 public constant COST_REGISTER = 1000000000000000000000;
```

CVF-1.240. FIXED

- **Category** Readability
- **Source** PolicyProposals.sol

Recommendation This value could be rendered as “800e18” or “8e20” or “0.8e21”.

```
74 uint256 public constant REFUND_IF_LOST = 8000000000000000000;
```

CVF-1.241. FIXED

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

Recommendation The type of this variable should be “PolicyVotes”.

```
88 address public policyVotesImpl;
```

CVF-1.242. INFO

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

Recommendation These variables should be declared as “immutable”.

Client Comment *Making this immutable requires redeploying this contract if a change is made to any of these files. To allow for easier governance, this is left mutable.*

```
88 address public policyVotesImpl;
```

```
93 address public simplePolicyImpl;
```

CVF-1.243. FIXED

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

Recommendation The type of this variable should be “SimplePolicySetter”.

```
93 address public simplePolicyImpl;
```

CVF-1.244. FIXED

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

Recommendation Events are usually named via nouns, such as “Proposal”, “VotingStart”, “ProposalSupporter”, “ProposalRefund”.

```
99  event ProposalAdded(address proposer, address proposalAddress);
105 event VotingStarted(address contractAddress);
111 event ProposalSupported(address supporter, address proposalAddress);
117 event ProposalRefunded(address proposer);
```

CVF-1.245. FIXED

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

Recommendation All parameters should be indexed.

```
99  event ProposalAdded(address proposer, address proposalAddress);
105 event VotingStarted(address contractAddress);
111 event ProposalSupported(address supporter, address proposalAddress);
117 event ProposalRefunded(address proposer);
```

CVF-1.246. FIXED

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

Recommendation The type of this argument should be “PolicyVotes”.

```
130 address _policyvotes,
```

CVF-1.247. FIXED

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

Recommendation The type of this argument should be “SimplePolicySetter”.

```
131 address _simplepolicy
```

CVF-1.248. FIXED

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

Recommendation These assignments would not be necessary if the “policyVotesImpl” and “simplePolicyImpl” variables were declared as “immutable”.

Client Comment *Making this immutable requires redeploying this contract if a change is made to any of these files. To allow for easier governance, this is left mutable.*

```
147 policyVotesImpl = PolicyProposals(_self).policyVotesImpl();
    simplePolicyImpl = PolicyProposals(_self).simplePolicyImpl();
```

CVF-1.249. FIXED

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

Description This function may return arbitrary large arrays and doesn’t scale.

Recommendation Consider implementing some way to get the length of the array and a certain range of elements from it.

Client Comment *A paginated response function was added to replace this one.*

```
156 function allProposalAddresses() public view returns (address[]
      ↪ memory) {
```

CVF-1.250. FIXED

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

Recommendation The argument type should be “Proposal”.

```
170 function registerProposal(address _prop) external {
```

CVF-1.251. FIXED

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

Description This function should return the index of the new proposal in the “allProposals” array.

```
170 function registerProposal(address _prop) external {
```

CVF-1.252. FIXED

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

Recommendation This assignment should be performed right before the first use of “_p”.

```
171 Props storage _p = proposals[_prop];
```

CVF-1.253. FIXED

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

Recommendation This check should be performed at the very beginning of the function.

```
173 require(_prop != address(0), "The proposal address can't be 0");
```

CVF-1.254. FIXED

- **Category** Readability
- **Source** PolicyProposals.sol

Recommendation This line could be simplified using the “+=” operator.

```
192 totalproposals = totalproposals + 1;
```

CVF-1.255. FIXED

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

Recommendation These assignments should be done right before the first usages of the corresponding variables.

```
207 uint256 _amount = votingPower(msg.sender, blockNumber);
    uint256 _total = totalVotingPower(blockNumber);
```

```
210 Props storage _p = proposals[address(_prop)];
```

CVF-1.256. FIXED

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

Recommendation The values “30” and “100” should be named constants.

```
239 if (_p.totalstake > (_total * 30) / 100) {
```

CVF-1.257. FIXED

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

Description Deploying a contract just to use it once is waste of gas.

Recommendation Consider implementing an ability to pass additional data through the “internalCommand” function to the called smart contract, so one smart contract could be used many times with different arguments.

```
243 SimplePolicySetter sps = SimplePolicySetter(
        SimplePolicySetter(simplePolicyImpl).clone(
            ID_POLICY_VOTES,
            address(pv)
        )
    );
    Policy(policy).internalCommand(address(sps));
```

CVF-1.258. FIXED

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

Description The contract deployed here is used only once.

Recommendation Consider implementing an ability to destroy it after use.

```
243 SimplePolicySetter sps = SimplePolicySetter(
        SimplePolicySetter(simplePolicyImpl).clone(
            ID_POLICY_VOTES,
            address(pv)
        )
    );
```

CVF-1.259. FIXED

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

Recommendation The conversions are redundant, as “policy” is already “address”.

```
306 address(uint160(policy)),
```

CVF-1.260. FIXED

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

Recommendation The type of this variable should be “CurrencyGovernance”.

```
24 address public bordaImpl;
```

CVF-1.261. INFO

- **Category** Procedural
- **Source** CurrencyTimer.sol

Recommendation These variables should be declared as “immutable”.

Client Comment *Making this immutable requires redeploying this contract if a change is made to any of these files. To allow for easier governance, this is left mutable.*

```
24 address public bordaImpl;
```

```
26 address public inflationImpl;
   address public lockupImpl;
```

```
29 address public simplePolicyImpl;
```

CVF-1.262. FIXED

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

Recommendation The type of this variable should be “Inflation”.

```
26 address public inflationImpl;
```

CVF-1.263. FIXED

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

Recommendation The type of this variable should be “Lockup”.

```
27 address public lockupImpl;
```

CVF-1.264. FIXED

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

Recommendation The type of this variable should be “SimplePolicySetter”.

```
29 address public simplePolicyImpl;
```

CVF-1.265. FIXED

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

Recommendation The value type for this mapping should be “Lockup”.

```
34 mapping(uint256 => address) public override lockups;
```

CVF-1.266. FIXED

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

Recommendation The key type for this mapping should be “Lockup”.

```
35 mapping(address => bool) public isLockup;
```

CVF-1.267. FIXED

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

Recommendation Events are usually named via nouns, such as “CurrencyGovernanceDecisionStart”.

```
37 event CurrencyGovernanceDecisionStarted(address contractAddress);
   event InflationStarted(address indexed addr);
   event LockupOffered(address indexed addr);
```

CVF-1.268. FIXED

- **Category** Readability
- **Source** CurrencyTimer.sol

Recommendation The argument should be indexed.

```
37 event CurrencyGovernanceDecisionStarted(address contractAddress);
```

CVF-1.269. FIXED

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

Recommendation The types of these arguments should be “CurrencyGovernance”, “Inflation”, “Lockup”, and “SimplePolicySetter” respectively.

```
43 address _borda,
   address _inflation,
   address _lockup,
   address _simplepolicy
```

CVF-1.270. FIXED

- **Category** Suboptimal
- **Source** CurrencyTimer.sol

Recommendation These assignments wouldn’t be necessary in case the variables would be declared as “immutable”.

Client Comment *Making this immutable requires redeploying this contract if a change is made to any of these files. To allow for easier governance, this is left mutable.*

```
57 bordaImpl = CurrencyTimer(_self).bordaImpl();
   inflationImpl = CurrencyTimer(_self).inflationImpl();
   lockupImpl = CurrencyTimer(_self).lockupImpl();
60 simplePolicyImpl = CurrencyTimer(_self).simplePolicyImpl();
```

CVF-1.271. FIXED

- **Category** Suboptimal
- **Source** CurrencyTimer.sol

Description Deploying a new contract just to use it once is waste of gas.

Recommendation Consider implementing an ability to pass additional information to the called contract through the “internalCommand” function so make it possible reusing the same contract.

```
95  SimplePolicySetter sps = SimplePolicySetter(
        SimplePolicySetter(simplePolicyImpl).clone(
            ID_CURRENCY_GOVERNANCE,
            _clone
        )
100 );
    Policy(policy).internalCommand(address(sps));
```

CVF-1.272. FIXED

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

Description The contract deployed here is not destroyed after use.

Recommendation Consider implementing an ability to destroy it.

```
95  SimplePolicySetter sps = SimplePolicySetter(
        SimplePolicySetter(simplePolicyImpl).clone(
            ID_CURRENCY_GOVERNANCE,
            _clone
        )
100 );
```

CVF-1.273. FIXED

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

Description The semantics of this functions is unclear.

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

```
5 function generation() external view returns (uint256);
```

CVF-1.274. FIXED

- **Category** Readability
- **Source** ECOxLockup.sol

Recommendation The “source” parameter should be indexed.

```
20 event Deposit(address source, uint256 amount);
```

CVF-1.275. FIXED

- **Category** Readability
- **Source** ECOxLockup.sol

Recommendation The “destination” parameter should be indexed.

```
28 event Withdrawal(address destination, uint256 amount);
```

CVF-1.276. FIXED

- **Category** Suboptimal
- **Source** ECOxLockup.sol

Description This check is redundant. Due to the net gas metering rules, it would be cheaper to always assign a value to a storage variable, regardless of whether this value is the same as the current one or not.

```
104 require(_new != _old, "Generation has not increased");
```

CVF-1.277. FIXED

- **Category** Procedural
- **Source** EcoTokenInit.sol

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

```
21 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.278. FIXED

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

Description Hardcoding addresses is a bad practice that makes testing harder.

Recommendation Consider passing the addresses array as a function or constructor argument.

```
29 //     0x79599DE87c2000b6aF219B37f7941E1Ab9b8E2d2,
33 //     0x6bAB1BD10Aa94431FF5d5bad537C93fCC2A78843,
37 //     0x6BFA48796115DFBA3005d5e14A0d1776cA4143c2,
41 //     0x39Be2F0d94b8a1f6a3d4FFFB996C6EDb62AF675f,
45 //     0x4D82E68a5e25C59Fa2A394676Ff309863E102dFA,
49 //     0x3aC84Fd4f17606811d3Dd12A7Af9329DC7f5597A,
53 //     0x386F1995345CA934b5121aF371314E677744e294,
57 //     0x4A2F5aDc7B0d732ea69258BcF19c22b00d8909e7,
61 //     0x5Dd3c3974256df9Ff663aCeF247A7702f79F5db3,
65 //     0xaa4d88B87d99D68079bfE87b089877FE54eE78a1,
69 //     0xE1A0d7F607c40e44A6E8908F2beC0f2080a1Eb11,
73 //     0x5db628E7Fa3C45E141c9292Cb2a5BbA12838fBea,
77 //     0xca73c3de68578Bf1805602ac5B020F37A7df0746,
81 //     0x7f9a3C42201Ee1914D28a1A4bAE6F7929f8F79d6,
85 //     0x70D661Ea6F8D0889f621EEAd54468D9629787D93,
89 //     0x8AB4153427C627FACfD27339fC12ce418ef98BE2,
93 //     0xF77E4a6382950b1b229f3767c87422eD9056AD30,
```

CVF-1.279. FIXED

- **Category** Procedural
- **Source** Policed.sol

Recommendation This contract should be declared as “abstract” as it is not supposed to be deployed as is, but should be inherited and extended.

```
11 contract Policed is ForwardTarget, IERC1820Implementer {
```

CVF-1.280. FIXED

- **Category** Suboptimal
- **Source** Policed.sol

Recommendation This assignment wouldn’t be necessary if “policy” would be an immutable variable.

```
71 policy = Policed(_self).policy();
```

CVF-1.281. INFO

- **Category** Readability
- **Source** Policed.sol

Description Most parts of this assembly bock could be written in plain Solidity.

Recommendation Consider using assembly only for those parts that cannot be written in Solidity, namely for returning and reverting with raw return data.

Client Comment *This assembly pattern is common (proxy calls for example) which makes it more readable as it’s recognizable. Assembly is retained to save gas.*

```
90 assembly {
```

CVF-1.282. FIXED

- **Category** Procedural
- **Source** EcoTestCleanup.sol

Recommendation This interface should be moved to a separate file named "Destructable.sol".

Client Comment *this file is test-only and is in the wrong folder and should not be considered*

```
13 interface Destructable {
```

CVF-1.283. FIXED

- **Category** Documentation
- **Source** EcoTestCleanup.sol

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

Client Comment *this file is test-only and is in the wrong folder and should not be considered*

```
27 constructor(address _policy) Policed(_policy) {}
```

CVF-1.284. FIXED

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

Recommendation The type of the "_target— argument should be "Destructable".

Client Comment *this file is test-only and is in the wrong folder and should not be considered*

```
38 function cleanup(address _target) external onlyOwner {
```

CVF-1.285. INFO

- **Category** Procedural
- **Source** EcoInitializable.sol

Recommendation This variable should be declared as immutable.

Client Comment *it is mutated in the function in this contract, it needs to be mutable*

```
23 address payable public owner;
```

CVF-1.286. INFO

- **Category** Suboptimal
- **Source** EcoInitializable.sol

Recommendation These assignments wouldn't be necessary if the "owner" variable would be immutable.

Client Comment *this is a mutating of the owner variable, it is set in the constructor, then mutated here.*

```
39 owner = payable(address(0));
```

```
59 owner = EcoInitializable(_self).owner();
```

CVF-1.287. FIXED

- **Category** Documentation
- **Source** EcoFaucet.sol

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

```
17 constructor(address _policy) PolicedUtils(_policy) {}
```

CVF-1.288. FIXED

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

Recommendation This value should be a constructor argument rather than a compile-ime constant.

```
23 uint256 internal constant NUM_PLACEHOLDERS = 20;
```

CVF-1.289. FIXED

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

Recommendation This value should be a named constant.

```
25 0xf86c915dad5894faca0dfa067c58fdf4307406d255ed0a65db394f82b77f53d4,
```

```
55 0xf86c915dad5894faca0dfa067c58fdf4307406d255ed0a65db394f82b77f53d4
```

CVF-1.290. FIXED

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

Recommendation Solidity does support using named constants inside assembly blocks.

```
55 0xf86c915dad5894faca0dfa067c58fdf4307406d255ed0a65db394f82b77f53d4
```

CVF-1.291. FIXED

- **Category** Suboptimal
- **Source** ForwardProxy.sol

Description This code the same for both branches.

Recommendation Consider coding it once before the conditional statement: 1. Execute “delegatecall“ and store the result in a local variable 2. Execute “returndatacopy” 3. Depending on the stored result, perform either “return” or “revert”.

```
64      returndatacopy(0x0, 0, returndatasize())
```

```
70  returndatacopy(0x0, 0, returndatasize())
```

CVF-1.292. FIXED

- **Category** Suboptimal
- **Source** PolicyInit.sol

Recommendation It would be more efficient to pass a single array of structs with two fields instead of two parallel arrays. This approach would also make the range check unnecessary.

```
29  bytes32[] calldata _keys,
30  address[] calldata _values,
```

CVF-1.293. FIXED

- **Category** Suboptimal
- **Source** PolicyInit.sol

Description The expression “_tokenResolvers[i]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
54  _tokenResolvers[i]
```

```
68  _tokenResolvers[i],
```

CVF-1.294. FIXED

- **Category** Procedural
- **Source** ForwardTarget.sol

Recommendation This contract should be declared as “abstract” as it is not supposed to be deployed as is, but should be inherited and extended.

8
```
contract ForwardTarget {
```

CVF-1.295. FIXED

- **Category** Suboptimal
- **Source** ForwardTarget.sol

Description This check is redundant.

Recommendation Just use the hash expression to initialize the “IMPLEMENTATION_SLOT” constant.

22
```
require(
    IMPLEMENTATION_SLOT ==
        uint256(
            keccak256(abi.encodePacked("com.eco.ForwardProxy.target"
                ↪ ))
        ),
    "IMPLEMENTATION_SLOT hash mismatch"
);
```

CVF-1.296. FIXED

- **Category** Suboptimal
- **Source** ForwardTarget.sol

Recommendation These variables are redundant, as Solidity allows using constant inside assembly blocks.

51
```
uint256 _sslot = IMPLEMENTATION_SLOT;
```

60
```
uint256 _sslot = IMPLEMENTATION_SLOT;
```

CVF-1.297. FIXED

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

Recommendation This contract could be turned into a library.

Client Comment *was better suited as abstract*

```
8 contract ERC1820Client {
```

CVF-1.298. INFO

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

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

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

Client Comment *ERC-1820 uses a nick's method deploy to always deploy to the same address. The constructor argument would be required in every single contract and passed up to the very top here. This solution more accurately reflects the reality of the implementation.*

```
10 IERC1820Registry(0x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24);
```

CVF-1.299. FIXED

- **Category** Suboptimal
- **Source** Policy.sol

Description This check is quite expensive and is performed for every setter until the desired setter is found, This could consume lots of gas.

Recommendation Consider allowing the transaction sender to provide a hint for the setter index to start looking at. This hint could be provided through a side-channel, e.g. in the lowest bits of "tx.gasprice". This would allow transaction sender to optimize gas costs by making this loop to be executed only once.

```
67 ERC1820REGISTRY.getInterfaceImplementer(
       address(this),
       setters[i]
70 ) == msg.sender
```

CVF-1.300. FIXED

- **Category** Suboptimal
- **Source** Policy.sol

Description There is no way to pass any additional arguments to the contract called here, while for some use cases additional arguments could be handy.

Recommendation Consider implementing an ability pass additional arguments.

Client Comment *This function is for use in Proposals where there is to be no runtime ambiguity.*

```
74 abi.encodeWithSignature("enacted(address)", _delegate)
```

CVF-1.301. FIXED

- **Category** Suboptimal
- **Source** Policy.sol

Description This function is redundant, as it just calls the inherited function with the same signature and modifiers.

Recommendation Consider removing this function.

```
96 function initialize(address _self)
```

CVF-1.302. INFO

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

Description The contract name looks like the name of a library, while actually this is a contract.

Recommendation Consider renaming to something like “PolicedBase”.

Client Comment *This function describes the utility helper functions that allow the based Policed object to work within our specific currency framework.*

```
15 contract PolicedUtils is Policed, CloneFactory, ERC1820Client {
```

CVF-1.303. FIXED

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

Recommendation These comments would be redundant if the hash expressions would be used to initialize constants instead of hardcoded values.

```
16 // keccak256("Faucet")
20 // keccak256("ERC20Token")
24 // keccak256("ContractCleanup")
28 // keccak256("TimedPolicies")
32 // keccak256("TrustedNodes")
36 // keccak256("PolicyProposals")
40 // keccak256("PolicyVotes")
44 // keccak256("EcoLabs")
48 // keccak256("CurrencyGovernance")
52 // keccak256("CurrencyTimer")
56 // keccak256("ECOx")
60 // keccak256("ECOxLockup")
```

CVF-1.304. FIXED

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

Recommendation This contract should be declared as “abstract” as it is not supposed to be deployed as is, but should be inherited and extended.

```
15 contract PolicedUtils is Policed, CloneFactory, ERC1820Client {
```

CVF-1.305. FIXED

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

Description These constants will be publicly available in all the contracts inherited from this contract. Public constants make a contract more expensive to deploy and use.

Recommendation Consider making these constants “internal”.

```
17 bytes32 public constant ID_FAUCET =
21 bytes32 public constant ID_ERC20TOKEN =
25 bytes32 public constant ID_CLEANUP =
29 bytes32 public constant ID_TIMED_POLICIES =
33 bytes32 public constant ID_TRUSTED_NODES =
37 bytes32 public constant ID_POLICY_PROPOSALS =
41 bytes32 public constant ID_POLICY_VOTES =
45 bytes32 public constant ID_ECO_LABS =
49 bytes32 public constant ID_CURRENCY_GOVERNANCE =
53 bytes32 public constant ID_CURRENCY_TIMER =
57 bytes32 public constant ID_ECOX =
61 bytes32 public constant ID_ECOXLOCKUP =
```

CVF-1.306. FIXED

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

Description The message is misleading, as "_addr" is not who calls this function.

Recommendation Consider rephrasing the message.

80
```
require(
    _addr == policy || _addr == expectedInterfaceSet,
    "Only the policy or interface contract may call this function."
);
```

CVF-1.307. FIXED

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

Description This function should emit some event.

107
```
function setExpectedInterfaceSet(address _addr) public onlyPolicy {
```

CVF-1.308. INFO

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

Description This function is redundant, as it is possible to clone this contract from outside. No need to code the clone logic into the contract itself.

Recommendation Consider removing this function.

Client Comment *This cloning pattern is one we wish to repeat in several places. This function is just a shorthand for that utility.*

123
```
function clone() public virtual returns (address) {
```

CVF-1.309. FIXED

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

Description This function allows cloning an existing clone (i.e. a proxy) rather than an implementation.

Recommendation Consider forbidding cloning a clone, or changing this function to always clone the implementation.

```
123 function clone() public virtual returns (address) {
```

CVF-1.310. FIXED

- **Category** Suboptimal
- **Source** TimeUtils.sol

Recommendation This contract could be turned into a library.

Client Comment *Made abstract instead*

```
7 contract TimeUtils {
```

CVF-1.311. FIXED

- **Category** Procedural
- **Source** TimeUtils.sol

Description It seems that for testing purposes this contract is replaced with another version that allows faking the current time. Another way to do this is to have an internal virtual function "getTime" in every contract that needs to know the current time. For production contracts this function just returns "block.timestamp". For testing purposes the production contracts are extended and this function is overridden to return a fake time, but also to check that the original function works correctly: contract A { function getTime () internal virtual view returns (uint) { return block.timestamp; } } contract ATest is A { uint private fakeTime; function setFakeTime (uint newFakeTime) public { fakeTime = newFakeTime; } function getTime () internal override view returns (uint) { require (super.getTime () == block.timestamp); return fakeTime; } }

Client Comment *While the time function mentioned here is useful, the comment is outdated and this function is actually used a decent amount in production and almost never in testing.*

```
10 * Used extensively in testing, but also useful in production for
```

CVF-1.312. FIXED

- **Category** Suboptimal
- **Source** CloneFactory.sol

Description Usually, an initialization function is called on a cloned contract right after cloning.

Recommendation Consider implementing a single function the clones a contract and the invokes the clone with given calldata.

Client Comment *The implementation of this function in PolicedUtils.sol does call the initialization function.*

```
34 function createClone(address target) internal returns (address
   ↪ result) {
```

CVF-1.313. FIXED

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

Description The semantics of this function is unclear from its signature.

Recommendation Consider giving descriptive names to the argument and the returned value and/or adding a documentation comment.

```
5 function lockups(uint256) external view returns (address);
```

CVF-1.314. FIXED

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

Description The interface name and the only function name inside this interface doesn't correlate to each other.

Recommendation Consider renaming the interface to something like "IGenerationNotifier".

```
4 interface ITimeNotifier {
      function notifyGenerationIncrease() external;
```

CVF-1.315. FIXED

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

Description The interface name is confusing, as it doesn't notify anybody, but rather is being notified itself.

Recommendation Consider renaming the interface to something like "ITimeNotifiable".

```
4 interface ITimeNotifier {
```

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

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting