Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

Blockchain

Security Audit Report

Project:	Smart Energy Chain
Domain:	smartenergychain.org
Platform:	Smart Energy Chain
Language:	Go And Solidity
Date:	November 27th, 2024

Contents

THIS IS A SECURITY AUDIT REPORT DOCUMENT THAT MAY CONTAIN INFORMATION THAT IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES THAT CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

About EtherAuthority

Founded in 2018 by blockchain experts and tech enthusiasts, EtherAuthority is a technology-led blockchain security company. Its mission is to prove the security and correctness of smart contracts and blockchain protocols through different approaches and detection methods, including manual, static, and dynamic analysis.

The official website is EtherAuthority.io and All the portfolio and public resources can be referred to at: https://github.com/etherauthority

The company's team of seasoned engineers and security auditors apply testing methodologies and verifications to ensure projects are checked against known attacks and potential vulnerabilities. To date, EtherAuthority has provided high-quality auditing and consulting services to 1000+ clients, including Catecoin, Smart Energy Chain, and HyperonChain.

The company customizes its engineering tool kits and applies cutting-edge research on smart contracts to each client's project to ensure high-quality delivery. As it continues to leverage technologies from blockchain and smart contracts, the EtherAuthority team will continue to support projects as a service provider and collaborator.

To verify the authenticity of this document, please refer to the official website, GitHub, or the social media announcements. Or, simply reach out to us: contact@etherauthority.io

Executive Summary

This report has been prepared for **Smart Energy Chain** to review the implementation, security, and soundness of their **Smart Energy Chain system.** A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Review the implementation and security of the **consensus** mechanism.
- Review the implementation and security of the **transaction** mechanism.
- Review the implementation and security of the **utxo** model.
- Review the **incentive** model.
- Review the **economic** model.
- Review the implementation of the delegated **proof of stake "DPos"** algorithm.

Review Summary

<table>
<tr><td rowspan="5">SECURITY LEVEL
 
</td><td colspan="2">This report has been prepared as a product of the Code Audit request by Smart Energy Chain.
This audit was conducted to discover issues and vulnerabilities in the source code of Smart Energy Chain.</td></tr>
<tr><td>TYPE</td><td>Smart Contract Platform</td></tr>
<tr><td>SOURCE CODE</td><td>https://github.com/secblockchain</td></tr>
<tr><td>GITHUB COMMIT</td><td>5e65b8a71f690fc22abad3ac4e2a27e63acae08e</td></tr>
<tr><td>ALGORITHM</td><td>Delegated proof of stake "DPos"</td></tr>
<tr><td rowspan="4"></td><td>PLATFORM</td><td>Smart Energy Chain</td></tr>
<tr><td>LANGUAGE</td><td>Go and Solidity</td></tr>
<tr><td>DELIVERY DATE</td><td>November 27th, 2024</td></tr>
<tr><td>METHODS</td><td>Dynamic Analysis, Static Analysis, and Manual Review have been performed.</td></tr>
</table>

Manual Review Notes

Introduction

The EtherAuthority team has been engaged by the Smart Energy Chain team to audit the design and implementation of its system. The audited source code links:
https://github.com/secblockchain/

Specifically, we examined the Git revisions for our initial review:
core-blockchain: 0cf89bdbb34aacb7fde3616ca05484ac6d2fad5f
system-smart-contracts: 5e65b8a71f690fc22abad3ac4e2a27e63acae08e

The goal of this audit is to review the Smart Energy Chain implementation of its core mechanisms general design and architecture, study potential security vulnerabilities, and uncover bugs that could compromise the software in production.

Documentation

We used the following sources with respect to our work:
1. Blockchain Domain: https://smartenergychain.org
2. Audit Scope: https://github.com/secblockchain

Project Background for Solidity Contracts

- Smart Energy Chain Smart Contracts handle multiple contracts, and all contracts have different functions.
    - Punish: This contract manages the punishment system for validators who miss blocks during their responsibilities. It tracks missed blocks and punishes validators based on thresholds. Validators may be removed or have their incoming validator status revoked after repeated missed blocks. The contract also supports reducing missed block counts periodically and cleaning up records of punished validators.
    - Proposal: This Solidity code represents a smart contract named Proposal for a decentralized governance system that involves creating and voting on proposals to appoint or reactivate validators. The contract ensures only

existing validators can vote, and tracks voting outcomes to either pass or reject a proposal.

- Validators: The contract includes functions for validators to stake/unstake, withdraw stake reward, withdraw profits, and distribute block reward to all active validators
- ValidatorData: This contract, ValidatorHelper, extends Ownable and interfaces with InterfaceValidator to manage staking operations related to validators, including creating/editing, unstaking, and withdrawing profits.

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the protocol implementation;
- Attack vectors related to building, running & maintaining the nodes (eg version upgrades, downloading new clients, fork notifications);
- User funds are secure on the blockchain and cannot be transferred without user permission;
- Vulnerabilities within each component as well as secure interaction between the network components;
- Correctly passing requests to the network core;
- Data privacy, data leaking, and information integrity;
- Key management implementation: secure private key storage and proper management of encryption and signing keys;
- Handling large volumes of network traffic;
- Resistance to DDoS and similar attacks;
- Aligning incentives with the rest of the network;
- Vulnerabilities, potential misuse, and gaming of smart contracts;
- Any attack that impacts funds, such as draining or manipulating of funds;
- Mismanagement of funds via transactions;
- Inappropriate permissions and excess authority;
- Secure communication between the nodes;
- Special token issuance model; and
- Anything else was identified during the initial analysis phase.

Claimed Smart Contract Features

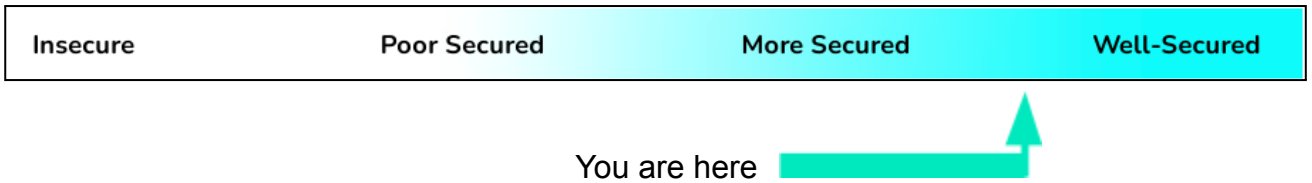
Claimed Feature Detail	Our Observation
File 1 Params.sol • Staker Part Percentage: 45% • Validator Part Percent: 30% • Burn Part Percent: 25% of every Transaction Fee • After 100 billion coins burn, it will stop burning • Minimal Staking Coin: 32 SEP • Min Validator Stack: 1 million SEP • Max Number of Validators: 21	**YES, This is valid.**
File 2 Punish.sol • The Miner has the ability to set a punishment address. • The Miner has the ability to decrease the value of the missed blocks counter. • The Validator Contract can be used to clean the punishment record if one stake in.	**YES, This is valid.**
File 3 Proposal.sol **Main Functions:** • **initialize**: Sets the initial validators and flags them as active. • **createProposal**: Allows users to propose a new validator or reactivation, provided no existing active proposal for the same address exists. • **voteProposal**: Casts a vote on an existing proposal by an active validator and updates the proposal's outcome based on majority voting. • **setUnpassed**: Marks a validator as not active.	**YES, This is valid.**

File 4 Validators.sol ● The new wallet allows a validator to use a specific amount of coins. ● Deligators can stake under active validators. ● The proposal contract owner is responsible for updating the validator status if the proposal is unstaked or jailed. ● The Miner exclusively distributes block rewards to all active validators. ● The Miner has the ability to update the active validator set. ● The Punish contract has the ability to remove the validator from the active set. ● The Punish contract has the ability to remove the validator's incoming address. **Owner Specifications:** ● Allows the contract owner to share the validator gain with the creator of the contract. ● Allows the owner to update gas settings. ● Allows the owner to update the parameter.	**YES, This is valid.**
File 5 ValidatorData.sol **Owner Specifications:** ● Allows the contract owner to withdraw all SEP from the contract. ● Allows the contract owner to change the minimum staking requirement.	**YES, This is valid.**

Audit Summary

The results of the review and automated tools along with the manual examination of the code bases provided several relevant findings regarding the application reviewed. The codebase in scope was mainly in **Go language and Solidity language**, as the project's chain, proof of stake algorithm, and VM, and a small part in **Javascript and C programming language** regarding the layer and functionality.

According to the standard audit assessment, the Customer`s solidity smart contracts are **"More Secured"**. This token contract does not have any ownership control, hence it is **100% decentralized**.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 0 very low-level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

AS-IS overview

Params.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyMiner	modifier	Passed	No Issue
3	onlyNotInitialized	modifier	Passed	No Issue
4	onlyInitialized	modifier	Passed	No Issue
5	onlyPunishContract	modifier	Passed	No Issue
6	onlyBlockEpoch	modifier	Passed	No Issue
7	onlyValidatorsContract	modifier	Passed	No Issue
8	onlyProposalContract	modifier	Passed	No Issue

Punish.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyNotPunished	modifier	Passed	No Issue
3	onlyNotDecreased	modifier	Passed	No Issue
4	initialize	external	access only Not Initialized	No Issue
5	punish	external	access only Miner	No Issue
6	decreaseMissedBlocksCounter	external	access only Miner	No Issue
7	cleanPunishRecord	write	access only Validators Contract	No Issue
8	getPunishValidatorsLen	read	Passed	No Issue
9	getPunishRecord	read	Passed	No Issue
10	onlyMiner	modifier	Passed	No Issue
11	onlyNotInitialized	modifier	Passed	No Issue
12	onlyInitialized	modifier	Passed	No Issue
13	onlyPunishContract	modifier	Passed	No Issue
14	onlyBlockEpoch	modifier	Passed	No Issue
15	onlyValidatorsContract	modifier	Passed	No Issue
16	onlyProposalContract	modifier	Passed	No Issue

Proposal.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyValidator	modifier	Passed	No Issue

3	initialize	external	access only Not Initialized	No Issue
4	createProposal	external	Passed	No Issue
5	voteProposal	external	access only Validator	No Issue
6	voteProposal	external	access only Validators Contract	No Issue
7	onlyMiner	modifier	Passed	No Issue
8	onlyNotInitialized	modifier	Passed	No Issue
9	onlyInitialized	modifier	Passed	No Issue
10	onlyPunishContract	modifier	Passed	No Issue
11	onlyBlockEpoch	modifier	Passed	No Issue
12	onlyValidatorsContract	modifier	Passed	No Issue
13	onlyProposalContract	modifier	Passed	No Issue

ValidatorData.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	receive	external	Passed	No Issue
3	createOrEditValidator	external	Passed	No Issue
4	unstake	external	Passed	No Issue
5	withdrawStakingReward	external	Passed	No Issue
6	viewValidatorRewards	read	Passed	No Issue
7	rescueCoins	external	access only Owner	No Issue
8	changeMinimumValidatorStaking	external	access only Owner	No Issue
9	getAllValidatorInfo	external	Passed	No Issue
10	validatorSpecificInfo1	external	Passed	No Issue
11	validatorSpecificInfo2	external	Passed	No Issue
12	waitingWithdrawProfit	external	Passed	No Issue
13	waitingUnstaking	external	Passed	No Issue
14	waitingWithdrawStaking	read	Passed	No Issue
15	minimumStakingAmount	external	Passed	No Issue
16	stakingValidations	external	Passed	No Issue
17	checkValidator	external	Passed	No Issue
18	onlyOwner	modifier	Passed	No Issue
19	owner	read	Passed	No Issue
20	_checkOwner	internal	Passed	No Issue
21	renounceOwnership	write	access only Owner	No Issue
22	transferOwnership	write	access only Owner	No Issue
23	_transferOwnership	internal	Passed	No Issue

Validators.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyNotRewarded	modifier	Passed	No Issue
3	onlyNotUpdated	modifier	Passed	No Issue
4	setContractCreator	write	Passed	No Issue
5	initialize	external	access only Not Initialized	No Issue
6	stake	write	access only Initialized	No Issue
7	createOrEditValidator	external	access only Initialized	No Issue
8	tryReactive	external	access only Proposal Contract	No Issue
9	unstake	external	access only Initialized	No Issue
10	withdrawStakingReward	write	Passed	No Issue
11	withdrawStaking	external	Passed	No Issue
12	withdrawProfits	external	Passed	No Issue
13	distributeBlockReward	external	access only Miner	No Issue
14	updateActiveValidatorSet	write	access only Miner	No Issue
15	removeValidator	external	access only Punish Contract	No Issue
16	removeValidatorIncoming	external	access only Punish Contract	No Issue
17	getValidatorDescription	read	Passed	No Issue
18	getValidatorInfo	read	Passed	No Issue
19	getStakingInfo	read	Passed	No Issue
20	getActiveValidators	read	Passed	No Issue
21	getTotalStakeOfActiveValidators	read	Passed	No Issue
22	getTotalStakeOfActiveValidators Except	read	Passed	No Issue
23	isActiveValidator	read	Passed	No Issue
24	isTopValidator	read	Passed	No Issue
25	getTopValidators	read	Passed	No Issue
26	validateDescription	write	Passed	No Issue
27	tryAddValidatorToHighestSet	internal	Passed	No Issue
28	tryRemoveValidatorIncoming	write	Passed	No Issue
29	addProfitsToActiveValidatorsByStakePercentExcept	write	Passed	No Issue
30	tryJailValidator	write	Passed	No Issue
31	tryRemoveValidatorInHighestSet	write	Passed	No Issue
32	viewStakeReward	read	Passed	No Issue
33	_updateGasSettings	internal	Passed	No Issue
34	updateGasSettings	external	access only Owner	No Issue
35	_updateParams	internal	Passed	No Issue

36	updateParams	external	access only Owner	No Issue
37	onlyOwner	modifier	Passed	No Issue
38	owner	read	Passed	No Issue
39	_checkOwner	internal	Passed	No Issue
40	renounceOwnership	write	access only Owner	No Issue
41	transferOwnership	write	access only Owner	No Issue
42	_transferOwnership	internal	Passed	No Issue
43	onlyMiner	modifier	Passed	No Issue
44	onlyNotInitialized	modifier	Passed	No Issue
45	onlyInitialized	modifier	Passed	No Issue
46	onlyPunishContract	modifier	Passed	No Issue
47	onlyBlockEpoch	modifier	Passed	No Issue
48	onlyValidatorsContract	modifier	Passed	No Issue
49	onlyProposalContract	modifier	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

General Comments

During our review, we found the Smart Energy Chain code to be well-structured and clear to follow. Smart Energy Chain demonstrates organized modular architecture, an effort to adhere to standardized development processes, and has adequate unit test coverage along with an open access test network environment - all of which are practices that are helpful for reducing the risk of security issues.

Smart Energy Chain also maintains a considerable amount of documentation and appears to proactively engage with the Smart Energy Chain community by maintaining an open source code base that supports independent security reviews, in addition to public communications channels, including Telegram, which encourage community participation. Both of these efforts help to encourage the discoverability of security issues and complement independent reviews like this one.

However, Smart Energy Chain's use of a delegated proof of stake "DPos" consensus algorithm exposes it to a host of potential attacks that have not yet been identified, largely due to DPos being a fairly new approach to achieving blockchain consensus. This exposure is inherent given that design and development of resilient systems are still in the early stages and have yet to be extensively researched. Systems that do run DPos in production face criticism for being too centralized, not addressing the "nothing-at-stake problem", and various complexities with regard to block reorganization.

Smart Energy Chain has taken reasonable steps to minimize the inherent risk with DPos, including soliciting an audit of their codebase and adding mechanisms like stacking and slashing. Precautions like stacking and slashing are incorporated to incentivize good behavior and discourage selfishness by network participants, however, it is not yet clear if these will be sufficient or stable throughout DPos network changes.

The use of DPos by Smart Energy Chain and other comparable DPos algorithms in other blockchains will likely require close observation and potentially require adjustments to the mechanisms in place over time.

Finally, simplification of core components of the Smart Energy Chain codebase by removing unused code should be a continuous effort. We acknowledge that such efforts are undertaken incrementally and iteratively, in order to increase code readability and allow for better review of code quality.

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No medium-severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

No very low-severity vulnerabilities were found.

Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

The following are Admin functions:

Validators.sol

- setContractCreator: This contract shares the validator gain with the creator of the contract.
- updateGasSettings: The owner can update gas settings.
- updateParams: The owner can update the parameter.

ValidatorHelper.sol

- rescueCoins: The owner can rescue coins.
- changeMinimumValidatorStaking: The owner can change the minimum validator staking amount.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Smart Energy Chain Implementation Analysis

Wallet Account Model

We have reviewed the account model. We have examined the related codebase, the modules used, and all related functionality and **we haven't found any discrepancies in the code.**

Consensus

The Smart Energy Chain consensus algorithm is a variant of the (DPos) Consensus. It aims to increase transaction processing throughput, decrease transaction confirmation latency, and enhance security by addressing the selfish-mining strategy.

All parts of the specification were found to be present and well translated to the code. **We have found it achieves these goals without introducing any new security issues.**

Incentive Model

The Smart Energy Chain incentive model is derived from its consensus. It shares some properties with the Polygon's BOR Consensus - namely miners are incentivized to mine blocks by issuing block rewards. **We have not found any game-theoretical issues in the Smart Energy Chain incentive model.**

Economic Model

The economic model is a subset of the incentive model of the system. The combination of the consensus model plus the Clique (DPos) algorithm creates a Smart Energy Chain security environment around the economic model.

Since the consensus model addresses some key problems found in many blockchains, the addition of a custom BOR algorithm from Polygon **hardens the model to an elevated security level. We have not found any issues arising out of the economics of the system.**

Test Cases and Coverage

Considering the fact that the team has provided evidence of stellar performance on the code design and implementation, the small percentage of code coverage does not reflect the overall performance of the tests.

Also, the code is forked from the Go-Ethereum 1.10.26 version, their test cases and coverages were considered automatically.

Conclusion

We were given a contract code in the form of a GitHub link and we have used all possible tests based on given objects. We have observed no issues in the smart contracts. **So it is good to go for the mainnet deployment.**

Since possible test cases can be unlimited for such blockchain smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

The blockchain code within the scope was manually reviewed and analyzed with static analysis tools. These tools provided mostly false positive results and thus they can be safely ignored.

The audit report contains all security vulnerabilities and other issues found in the reviewed code.

The security state of the reviewed blockchain code, based on standard audit procedure scope, is **“More Secured”.**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of the functionality of the software under review. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthoroty.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

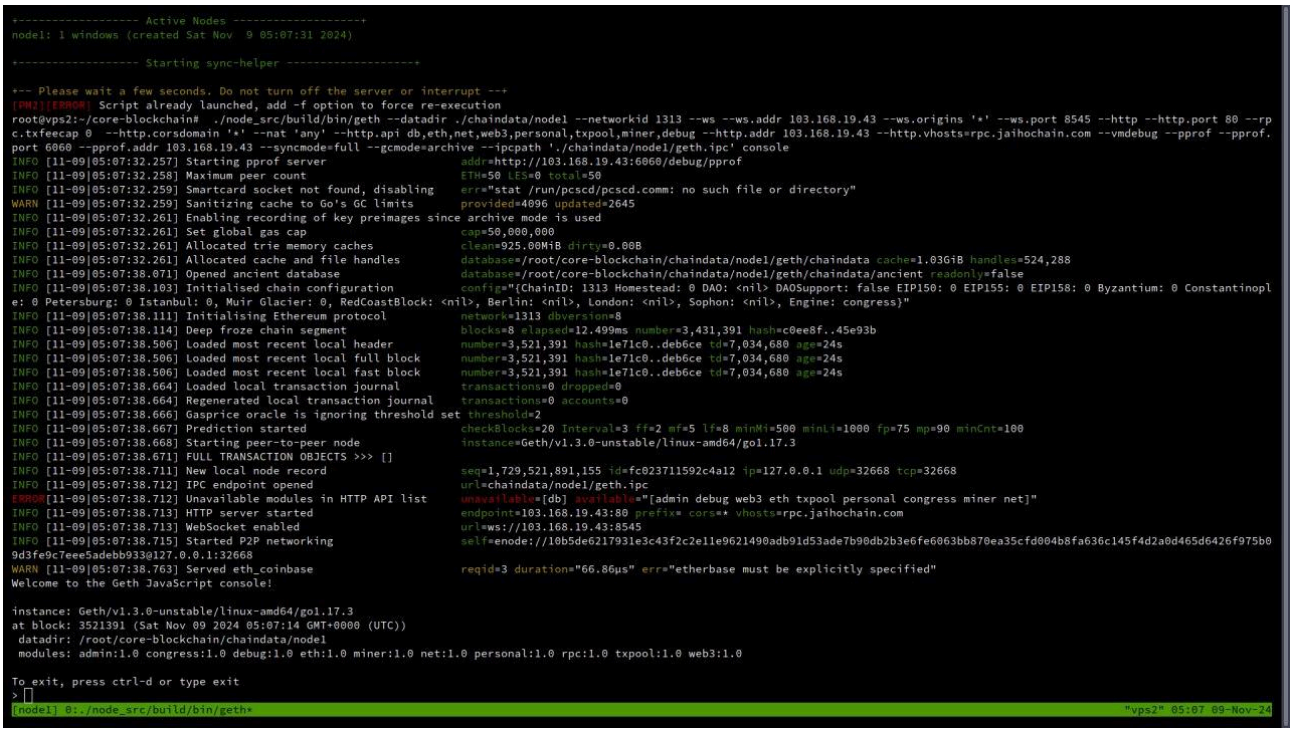
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

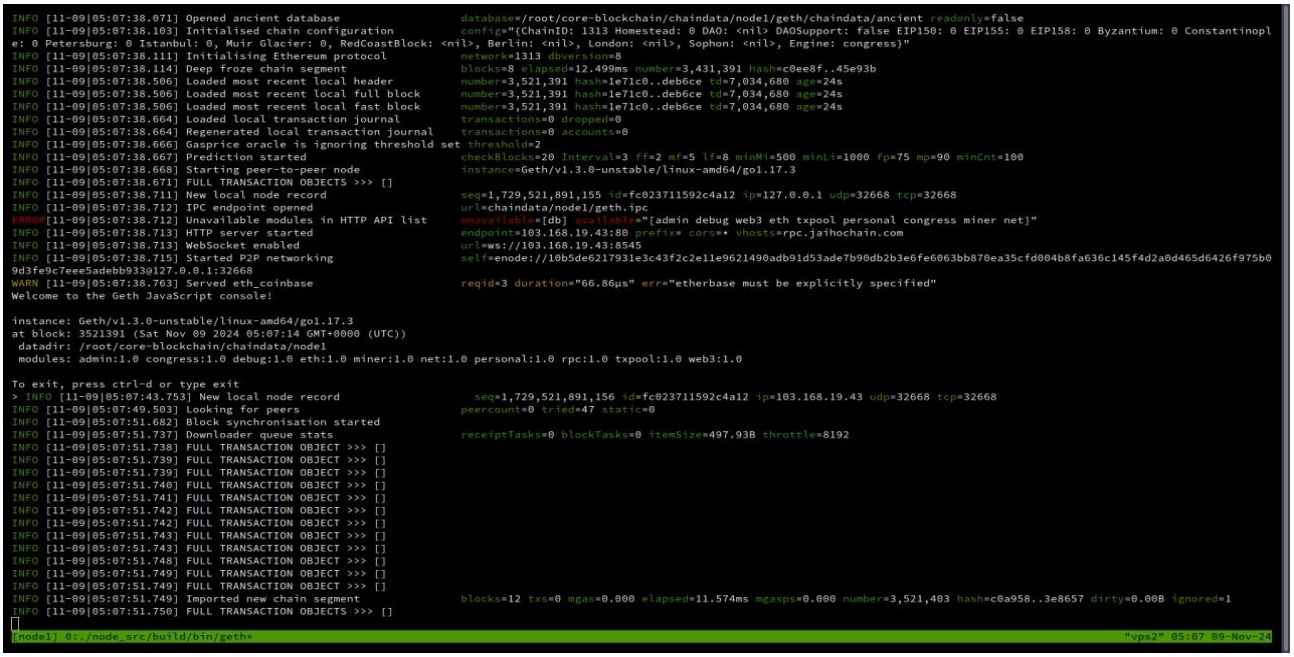
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

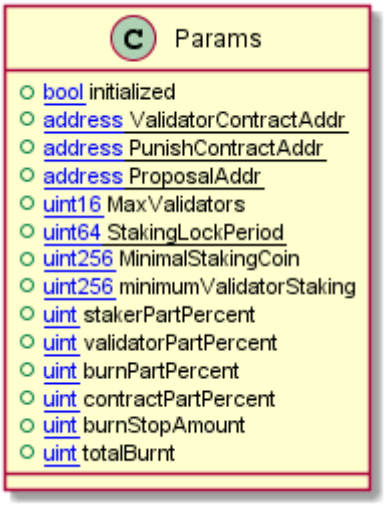
We compared the entire source code of Smart Energy Chain Chain with go-ethereum source version 1.10.26. And we inspected differences manually for each change. Then we compiled with Go version 1.17 and tested by running geth, which worked successfully in all aspects. Please find the screenshot below for the same.



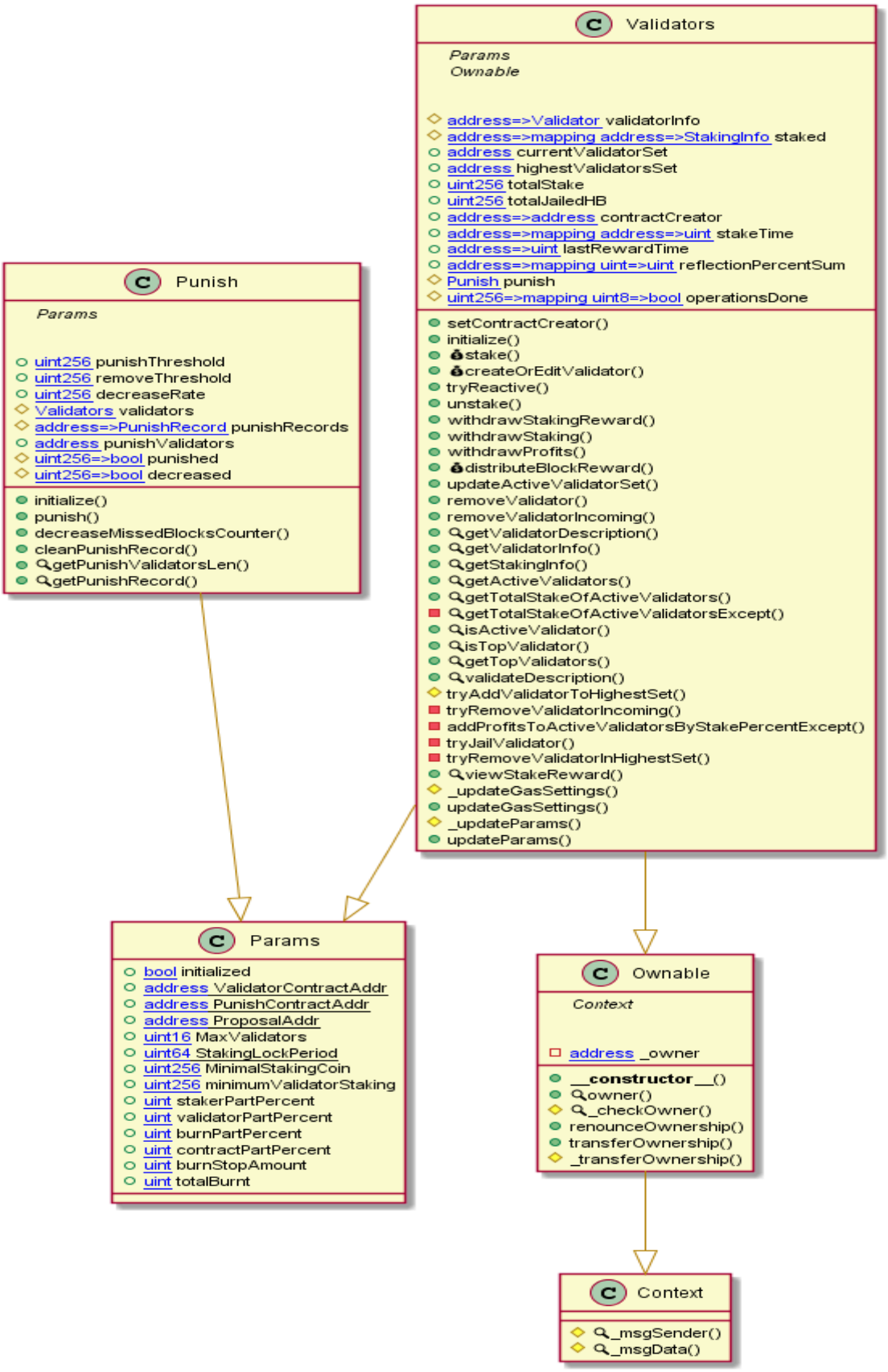


Code Flow Diagram - Smart Energy Chain Protocol

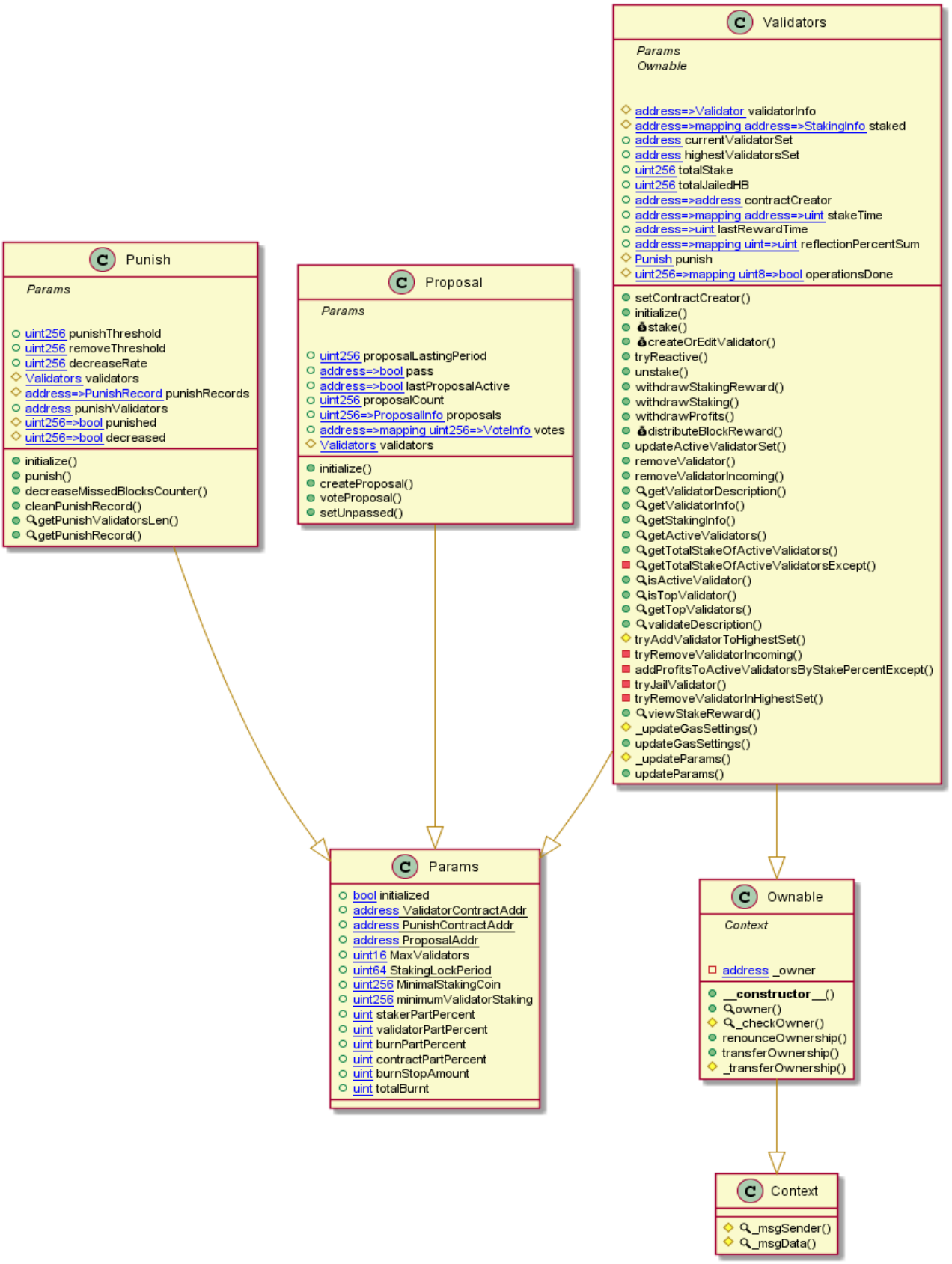
Params Diagram



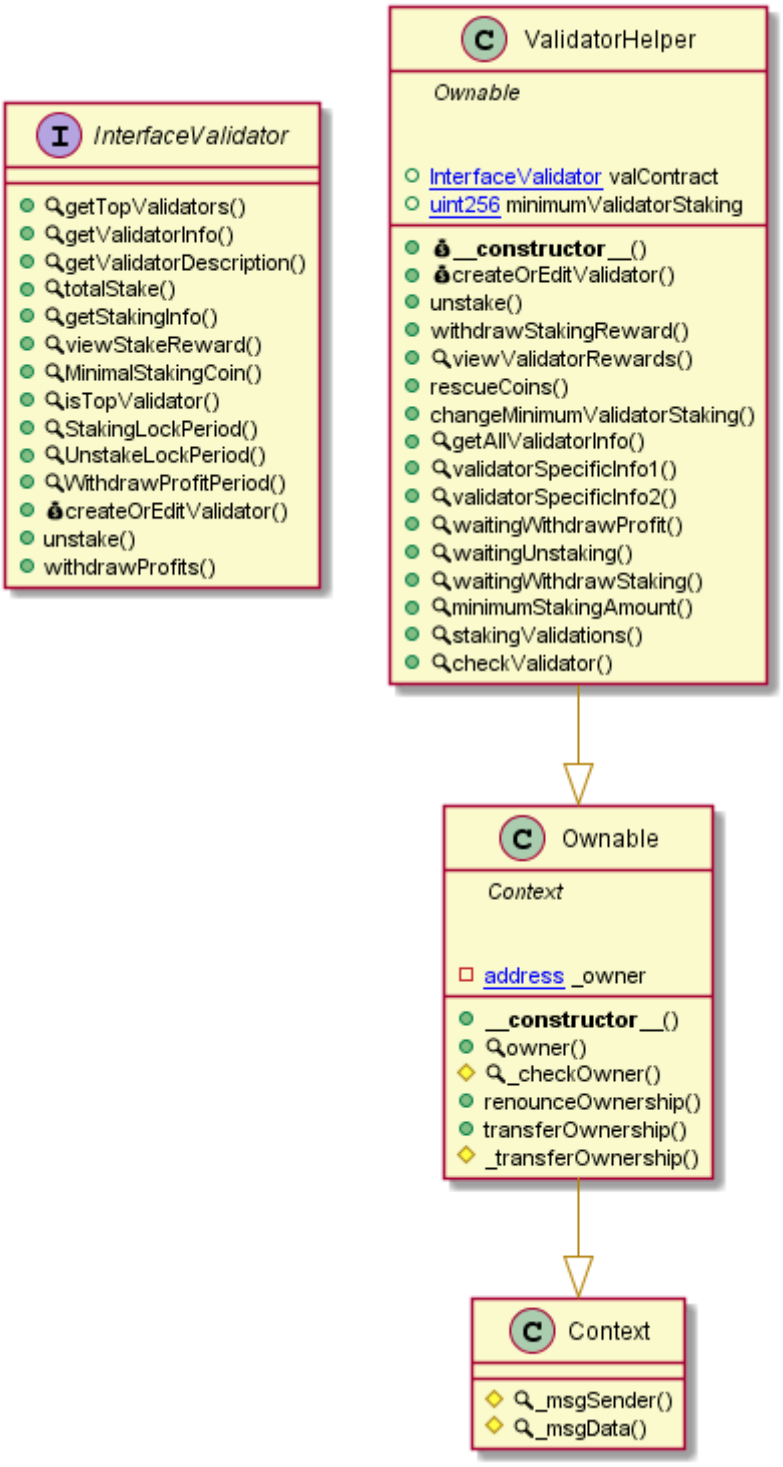
Punish Diagram



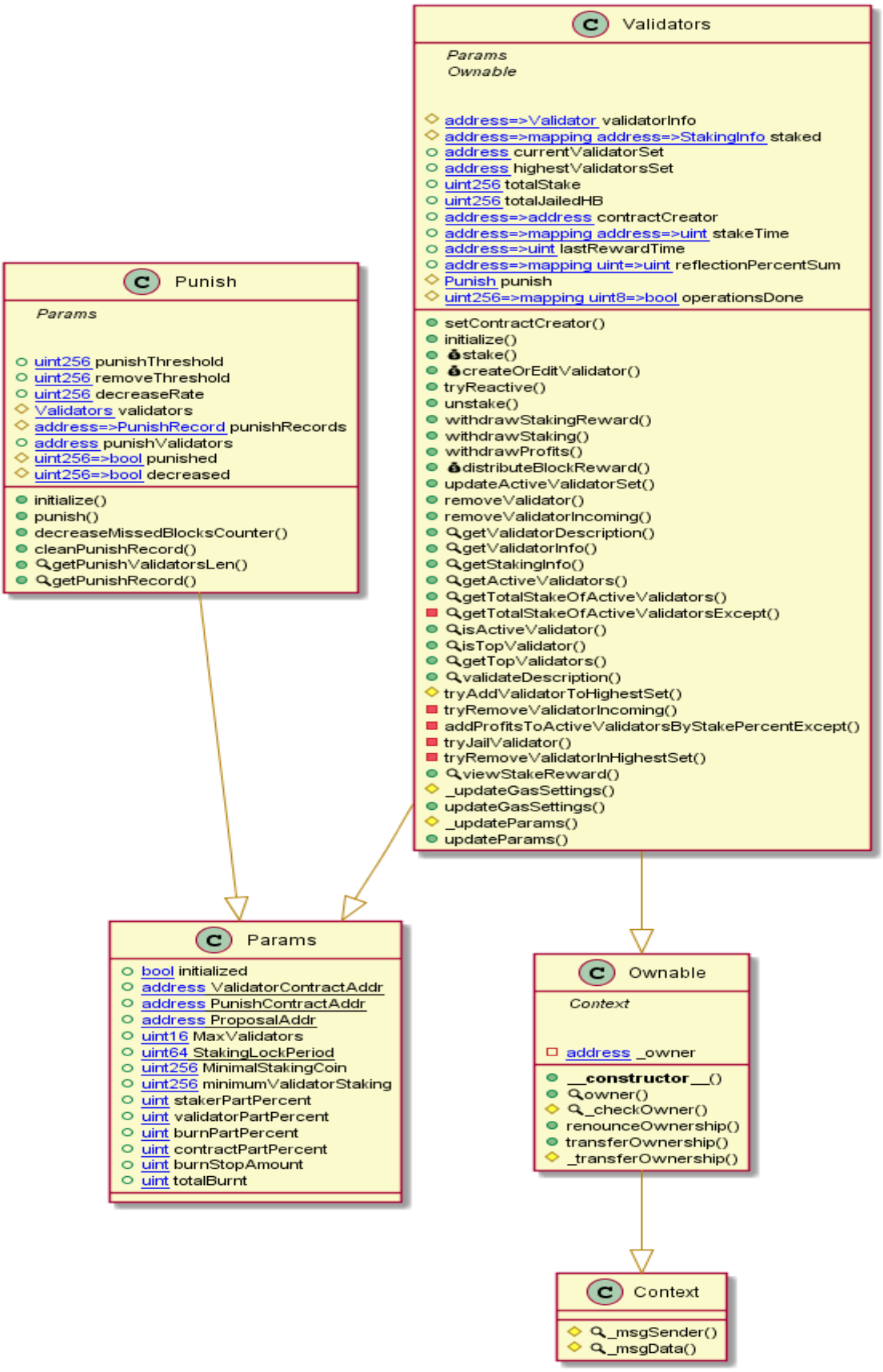
Proposal Diagram



ValidatorData Diagram



Validators Diagram



Slither Results Log

Slither Log >> Params.sol

```
INFO:Detectors:
Params.initialized (Params.sol#5) is never initialized. It is used in:
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
Pragma version0.8.17 (Params.sol#2) allows old versions
solc-0.8.17 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Params.MaxValidators (Params.sol#16) is not in mixedCase
Variable Params.MinimalStakingCoin (Params.sol#21) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Params.slitherConstructorVariables() (Params.sol#4-73) uses literals with too many digits:
      - minimumValidatorStaking = 100000000000000000000000 (Params.sol#23)
Params.slitherConstructorVariables() (Params.sol#4-73) uses literals with too many digits:
      - burnStopAmount = 100000000000000000000000000000 (Params.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:Params.sol analyzed (1 contracts with 93 detectors), 7 result(s) found
```

Slither Log >> Proposal.sol

```
INFO:Detectors:
Proposal.createProposal(address,string) (Proposal.sol#1300-1321) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(proposals[id].createTime == 0,Proposal already exists) (Proposal.sol#1309)
Proposal.voteProposal(uint256,bool) (Proposal.sol#1323-1379) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(proposals[id].createTime != 0,Proposal not exist) (Proposal.sol#1328)
        - require(bool,string)(block.timestamp < proposals[id].createTime + proposalLastingPeriod,Proposal expired) (Proposal.sol#1333-1336)
        - proposals[id].reject >=
```

```
validators.getActiveValidators().length / 2 + 1
(Proposal.sol#1371-1372)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#block-t
imestamp
INFO:Detectors:
Pragma version0.8.17 (Proposal.sol#2) allows old versions
solc-0.8.17 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorre
ct-versions-of-solidity
INFO:Detectors:
Parameter
Validators.updateParams(uint16,uint256,uint256)._MaxValidators
(Proposal.sol#1213) is not in mixedCase
Parameter
Validators.updateParams(uint16,uint256,uint256)._MinimalStakingCoin
(Proposal.sol#1213) is not in mixedCase
Parameter
Validators.updateParams(uint16,uint256,uint256)._minimumValidatorStak
ing (Proposal.sol#1213) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conform
ance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in Validators.withdrawStakingReward(address)
(Proposal.sol#692-706):
        External calls:
        - address(tx.origin).transfer(reward) (Proposal.sol#702)
        Event emitted after the call(s):
        -
withdrawStakingRewardEv(tx.origin,validator,reward,block.timestamp)
(Proposal.sol#703)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentra
ncy-vulnerabilities-4
INFO:Detectors:
Proposal.slitherConstructorVariables() (Proposal.sol#1220-1392) uses
literals with too many digits:
        - minimumValidatorStaking = 10000000000000000000000
(Proposal.sol#23)
Proposal.slitherConstructorVariables() (Proposal.sol#1220-1392) uses
literals with too many digits:
        - burnStopAmount = 100000000000000000000000000000
(Proposal.sol#31)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#too-man
y-digits
INFO:Detectors:
Loop condition i < currentValidatorSet.length (Proposal.sol#953)
should use cached array length instead of referencing `length` member
of the storage array.
 Loop condition i < punishValidators.length (Proposal.sol#155) should
use cached array length instead of referencing `length` member of the
```

```
storage array.
 Loop condition i < highestValidatorsSet.length (Proposal.sol#1010)
should use cached array length instead of referencing `length` member
of the storage array.
 Loop condition i_scope_0 < highestValidatorsSet.length
(Proposal.sol#1025) should use cached array length instead of
referencing `length` member of the storage array.
 Loop condition i < currentValidatorSet.length (Proposal.sol#1097)
should use cached array length instead of referencing `length` member
of the storage array.
 Loop condition i_scope_0 < currentValidatorSet.length
(Proposal.sol#1120) should use cached array length instead of
referencing `length` member of the storage array.
 Loop condition i < highestValidatorsSet.length (Proposal.sol#977)
should use cached array length instead of referencing `length` member
of the storage array.
 Loop condition i < currentValidatorSet.length (Proposal.sol#967)
should use cached array length instead of referencing `length` member
of the storage array.
 Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#cache-a
rray-length
INFO:Slither:Proposal.sol analyzed (6 contracts with 93 detectors),
79 result(s) found
```

Slither Log >> Punish.sol

```
INFO:Detectors:
Reentrancy in Punish.punish(address) (Punish.sol#1128-1153):
	External calls:
	- validators.removeValidator(val) (Punish.sol#1143)
	- validators.removeValidatorIncoming(val) (Punish.sol#1149)
	Event emitted after the call(s):
	- LogPunishValidator(val,block.timestamp) (Punish.sol#1152)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Pragma version0.8.17 (Punish.sol#2) allows old versions
solc-0.8.17 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter Validators.updateParams(uint16,uint256,uint256)._MaxValidators (Punish.sol#1079)
is not in mixedCase
Parameter Validators.updateParams(uint16,uint256,uint256)._MinimalStakingCoin
(Punish.sol#1079) is not in mixedCase
Parameter Validators.updateParams(uint16,uint256,uint256)._minimumValidatorStaking
```

```
(Punish.sol#1079) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Punish.slitherConstructorVariables() (Punish.sol#1087-1218) uses literals with too many digits:
        - minimumValidatorStaking = 100000000000000000000000 (Punish.sol#23)
Punish.slitherConstructorVariables() (Punish.sol#1087-1218) uses literals with too many digits:
        - burnStopAmount = 10000000000000000000000000000 (Punish.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
 Loop condition i_scope_0 < currentValidatorSet.length (Punish.sol#986) should use cached array length instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Slither:Punish.sol analyzed (5 contracts with 93 detectors), 70 result(s) found
```

Slither Log >> ValidatorData.sol

```
INFO:Detectors:
ValidatorHelper.changeMinimumValidatorStaking(uint256) (ValidatorData.sol#235-237) should emit an event for:
        - minimumValidatorStaking = amount (ValidatorData.sol#236)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
ValidatorHelper.getAllValidatorInfo() (ValidatorData.sol#243-276) has external calls inside a loop: (status,coins) = valContract.getValidatorInfo(highestValidatorsSet[i]) (ValidatorData.sol#255)
ValidatorHelper.getAllValidatorInfo() (ValidatorData.sol#243-276) has external calls inside a loop: (identity,website) = valContract.getValidatorDescription(highestValidatorsSet[i]) (ValidatorData.sol#257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in ValidatorHelper.createOrEditValidator(address,string,string,string,string,string) (ValidatorData.sol#175-193):
        External calls:
        - valContract.createOrEditValidator{value: msg.value}(feeAddr,moniker,identity,website,email,details) (ValidatorData.sol#188)
        Event emitted after the call(s):
        - Stake(msg.sender,msg.value,block.timestamp) (ValidatorData.sol#190)
Reentrancy in ValidatorHelper.unstake(address) (ValidatorData.sol#196-204):
        External calls:
```

```
        - valContract.unstake(validator) (ValidatorData.sol#200)
        Event emitted after the call(s):
        - Unstake(msg.sender,block.timestamp) (ValidatorData.sol#202)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (ValidatorData.sol#72-74) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.17 (ValidatorData.sol#2) allows old versions
solc-0.8.17 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function InterfaceValidator.MinimalStakingCoin() (ValidatorData.sol#32) is not in mixedCase
Function InterfaceValidator.StakingLockPeriod() (ValidatorData.sol#34) is not in mixedCase
Function InterfaceValidator.UnstakeLockPeriod() (ValidatorData.sol#35) is not in mixedCase
Function InterfaceValidator.WithdrawProfitPeriod() (ValidatorData.sol#36) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ValidatorHelper.valContract (ValidatorData.sol#162) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:ValidatorData.sol analyzed (4 contracts with 93 detectors), 26 result(s) found
```

Slither Log >> Validators.sol

```
INFO:Detectors:
Validators.withdrawStakingReward(address) (Validators.sol#692-706) sends eth to arbitrary user
        Dangerous calls:
        - address(tx.origin).transfer(reward) (Validators.sol#702)
Validators.withdrawStaking(address) (Validators.sol#708-732) sends eth to arbitrary user
        Dangerous calls:
        - staker.transfer(staking) (Validators.sol#728)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Validators.addProfitsToActiveValidatorsByStakePercentExcept(uint256,address)
```

```
(Validators.sol#1071-1142) performs a multiplication on the result of a division:
	- per = totalReward / (rewardValsLen) (Validators.sol#1095)
	- remain = totalReward - (per * rewardValsLen) (Validators.sol#1096)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in Validators.tryReactive(address) (Validators.sol#609-631):
	External calls:
	- require(bool,string)(punish.cleanPunishRecord(validator),clean failed) (Validators.sol#624)
	State variables written after the call(s):
	- validatorInfo[validator].status = Status.Staked (Validators.sol#626)
	Validators.validatorInfo (Validators.sol#355) can be used in cross function reentrancies:
	- Validators.addProfitsToActiveValidatorsByStakePercentExcept(uint256,address)
(Validators.sol#1071-1142)
	- Validators.createOrEditValidator(address,string,string,string,string,string)
(Validators.sol#561-607)
	- Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#775-838)
	- Validators.getTotalStakeOfActiveValidatorsExcept(address) (Validators.sol#948-964)
	- Validators.getValidatorDescription(address) (Validators.sol#872-892)
	- Validators.getValidatorInfo(address) (Validators.sol#894-918)
	- Validators.initialize(address[]) (Validators.sol#468-493)
	- Validators.removeValidator(address) (Validators.sol#855-866)
	- Validators.stake(address) (Validators.sol#496-559)
	- Validators.tryAddValidatorToHighestSet(address,uint256) (Validators.sol#1006-1044)
	- Validators.tryJailValidator(address) (Validators.sol#1144-1155)
	- Validators.tryReactive(address) (Validators.sol#609-631)
	- Validators.tryRemoveValidatorIncoming(address) (Validators.sol#1046-1068)
	- Validators.unstake(address) (Validators.sol#633-690)
	- Validators.withdrawProfits(address) (Validators.sol#735-771)
	- Validators.withdrawStaking(address) (Validators.sol#708-732)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Validators.withdrawStakingReward(address) (Validators.sol#692-706) uses tx.origin for
authorization: require(bool,string)(stakeTime[tx.origin][validator] > 0,nothing staked)
(Validators.sol#694)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-usage-of-txorigin
INFO:Detectors:
Validators.addProfitsToActiveValidatorsByStakePercentExcept(uint256,address).last
(Validators.sol#1091) is a local variable never initialized
Validators.addProfitsToActiveValidatorsByStakePercentExcept(uint256,address).added
(Validators.sol#1119) is a local variable never initialized
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
```

```
Validators.withdrawStaking(address).staker (Validators.sol#709) lacks a zero-check on :
        - staker.transfer(staking) (Validators.sol#728)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#775-838) has external calls
inside a loop: address(contractCreator[_to[i]]).transfer(amt) (Validators.sol#811)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in Validators.tryReactive(address) (Validators.sol#609-631):
    External calls:
    - require(bool,string)(punish.cleanPunishRecord(validator),clean failed) (Validators.sol#624)
    Event emitted after the call(s):
    - LogReactive(validator,block.timestamp) (Validators.sol#628)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Validators.onlyNotRewarded() (Validators.sol#441-447) compares to a boolean constant:
    -require(bool,string)(operationsDone[block.number][uint8(Operations.Distribute)] ==
false,Block is already rewarded) (Validators.sol#442-445)
Validators.onlyNotUpdated() (Validators.sol#449-456) compares to a boolean constant:
    -require(bool,string)(operationsDone[block.number][uint8(Operations.UpdateValidators)] ==
false,Validators already updated) (Validators.sol#450-454)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (Validators.sol#224-226) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.17 (Validators.sol#2) allows old versions
solc-0.8.17 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter Validators.updateParams(uint16,uint256,uint256)._minimumValidatorStaking
(Validators.sol#1213) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Reentrancy in Validators.withdrawStakingReward(address) (Validators.sol#692-706):
    External calls:
    - address(tx.origin).transfer(reward) (Validators.sol#702)
    Event emitted after the call(s):
    - withdrawStakingRewardEv(tx.origin,validator,reward,block.timestamp)
(Validators.sol#703)
```

```
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Validators.slitherConstructorVariables() (Validators.sol#311-1218) uses literals with too many
digits:
        - burnStopAmount = 100000000000000000000000000000 (Validators.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
 Loop condition i < punishValidators.length (Validators.sol#155) should use cached array length
instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Slither:Validators.sol analyzed (5 contracts with 93 detectors), 70 result(s) found
```

Solidity Static Analysis

Params.sol

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 37:8:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 70:8:

Proposal.sol

Transaction origin:
Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
Pos: 300:41:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 102:30:

Gas costs:
Gas requirement of function Proposal.initialize is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 266:4:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at

maximum you can pass to such functions to make it successful.
Pos: 87:8:

Similar variable names:
Proposal.initialize(address[]) : Variables have very similar names "pass" and "vals". Note: Modifiers are currently not considered by this static analysis.
Pos: 79:32:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 116:8:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 590:25:

Punish.sol

Transaction origin:
Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
Pos: 262:37:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 402:54:

Gas costs:
Gas requirement of function Punish.decreaseMissedBlocksCounter is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 75:4:

Gas costs:
Gas requirement of function Validators.updateParams is infinite: If the gas requirement of a

function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1011:4:

Similar variable names:
Validators.stake(address) : Variables have very similar names "staked" and "staker". Note: Modifiers are currently not considered by this static analysis.
Pos: 355:22:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 492:8:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 983:23:

ValidatorData.sol

Transaction origin:
Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
Pos: 207:29:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 190:42:

Gas costs:
Gas requirement of function ValidatorHelper.viewValidatorRewards is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 214:4:

No return:
ValidatorHelper.validatorSpecificInfo2(address,address): Mixing of named and unnamed return parameters is not advised.
Pos: 297:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 186:8:

Validators.sol

Transaction origin:
Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
Pos: 581:22:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 970:53:

Gas costs:
Gas requirement of function Validators.initialize is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 266:4:

Gas costs:
Gas requirement of function Validators.withdrawStakingReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 490:4:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit

which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 269:8:

Similar variable names:
Validators. stake(address) : Variables have very similar names "staked" and "staker". Note: Modifiers are currently not considered by this static analysis.
Pos: 310:19:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 796:8:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for the division of (only) literal values since those yield rational constants.
Pos: 590:25:

Solhint Linter

Params.sol

```
Compiler version 0.8.17 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Constant name must be in capitalized SNAKE_CASE
Pos: 5:7
Constant name must be in capitalized SNAKE_CASE
Pos: 5:9
Constant name must be in capitalized SNAKE_CASE
Pos: 5:11
Variable name must be in mixedCase
Pos: 5:15
Constant name must be in capitalized SNAKE_CASE
Pos: 5:17
Variable name must be in mixedCase
Pos: 5:20
```

Punish.sol

```
Compiler version 0.8.17 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path ./Params.sol is not allowed. Specify names to
import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:3
global import of path ./Validators.sol is not allowed. Specify names
to import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:4
Explicitly mark visibility of state
Pos: 5:23
Avoid making time-based decisions in your business logic
Pos: 38:71
```

Proposal.sol

```
Compiler version 0.8.17 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path ./Params.sol is not allowed. Specify names to
import individually or bind all exports of the module into a name
```

```
(import "path" as Name)
Pos: 1:3
global import of path ./Validators.sol is not allowed. Specify names
to import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:4
Explicitly mark visibility of state
Pos: 5:43
Avoid making time-based decisions in your business logic
Pos: 31:101
Avoid making time-based decisions in your business logic
Pos: 53:104
Error message for require is too long
Pos: 9:115
Avoid making time-based decisions in your business logic
Pos: 59:161
Avoid making time-based decisions in your business logic
Pos: 34:175
```

ValidatorsData.sol

```
Compiler version 0.8.19 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Function name must be in mixedCase
Pos: 5:31
Error message for require is too long
Pos: 9:185
Avoid making time-based decisions in your business logic
Pos: 43:189
Avoid making time-based decisions in your business logic
Pos: 34:201
Avoid to use tx.origin
Pos: 30:206
Variable "user" is unused
Pos: 63:278
Variable "user" is unused
Pos: 36:306
Variable "validatorAddress" is unused
Pos: 50:306
Variable "user" is unused
Pos: 31:313
Variable "validator" is unused
Pos: 45:313
Variable "user" is unused
Pos: 29:346
```

Validators.sol

```
Compiler version 0.8.17 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path ./Params.sol is not allowed. Specify names to
import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:3
global import of path ./Punish.sol is not allowed. Specify names to
import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:4
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:49
Error message for require is too long
Pos: 9:91
Avoid making time-based decisions in your business logic
Pos: 31:615
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 13:618
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 13:622
Avoid making time-based decisions in your business logic
Pos: 54:969
Variable name must be in mixedCase
Pos: 28:1003
Variable name must be in mixedCase
Pos: 51:1003
Error message for require is too long
Pos: 7:1004
Use double quotes for string literals
Pos: 62:1004
Variable name must be in mixedCase
Pos: 27:1010
Variable name must be in mixedCase
Pos: 50:1010
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.