Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Vention
Website:	https://vention.app
Platform:	Vention Smart Chain
Language:	Solidity
Date:	February 27th, 2023

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH 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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Vention team to perform the Security audit of the The Vention Protocol smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on February 27th, 2023.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The Vention token (VNT) will run the marketplace and NFT creation platform that are part of the Vention ecosystem.
- In the Vention ecosystem, Vention tokens are required as payment for using the ecosystem's services.
- The system smart contracts contribute heavily to the consensus mechanism.
- The system smart contracts performs actions such as Validations, system staking, punishments, etc.

Audit scope

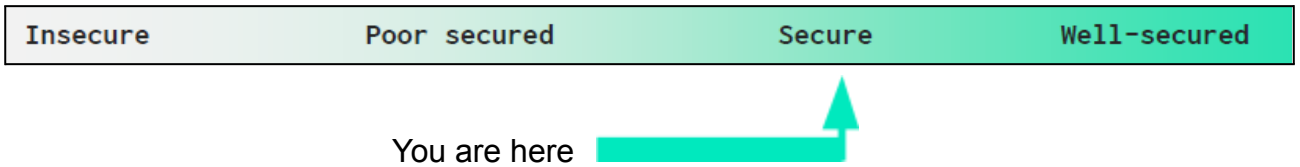
Name	**Code Review and Security Analysis Report for Vention Protocol Smart Contracts**
Platform	**Vention Smart Chain / Solidity**
File 1	Proposal.sol
File 2	Punish.sol
File 3	Validators.sol
Github Commit Hash	91cbc324ce35f1f44a3d828ae72daca8067348fc
Audit Date	February 27th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Token Specification: • Name: Vention • Symbol: VNT • Decimals: 18 • Total supply: 700 million	
File 1 Proposal.sol **Validator Specifications:** • The Owner can vote for proposals. • The Owner can set validator unpass addresses.	**YES, This is valid.**
File 2 Punish.sol **Miner Specifications:** • The owner can punish wallet addresses. • The owner can decrease missed block counter epoch. **Validator Specifications:** • The owner can cleanly punish record wallet addresses.	**YES, This is valid.**
File 3 Validators.sol **Proposal Specifications:** • : The owner can update the validator address whose status is Unstaked / Jailed.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. These contracts do contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 low and some very low level issues.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	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 not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 3 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in The Vention Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the The Vention Protocol.

The Vention Protocol team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **not** well commented on smart contracts.

Documentation

We were given The Vention Protocol smart contract code in the form of a github weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. But the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website: https://vention.app which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

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	setUnpassed	external	access only Validators Contract	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

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	getTotalStakeOfActiveValidatorsExcept	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

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 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 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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) data: **Params.sol**
data

Resolution: data

Very Low / Informational / Best practices:

Centralization

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

Proposal.sol

- voteProposal: Validators can vote for proposals.

- setUnpassed: Validators can set validator unpass addresses.

Punish.sol

- punish: The Miner owner can punish wallet addresses.
- decreaseMissedBlocksCounter: The Miner owner can decrease missed block counter epoch.
- cleanPunishRecord: Validators can cleanly punish record wallet addresses.

Validators.sol

- tryReactive: The owner of the proposal contract can update the validator address whose status is Unstaked / Jailed.

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

Conclusion

We were given a contract code in the form of a github weblink. And we have used all possible tests based on given objects as files. We have not observed any major issues in the smart contracts. So, **it's good to go to production**.

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

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

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

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.

Email: audit@EtherAuthority.io

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 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 whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. 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, 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

EtherAuthority.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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

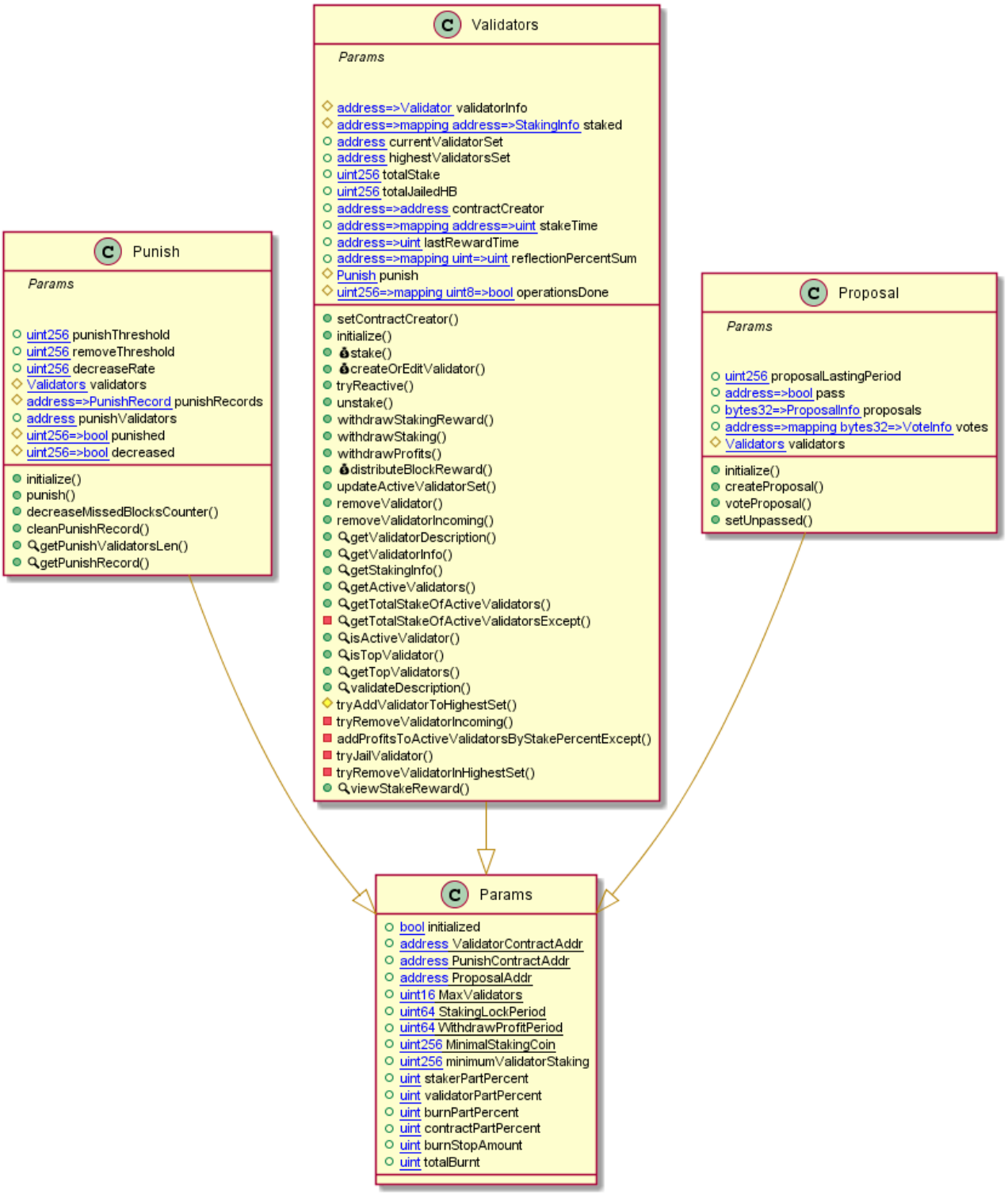
Technical Disclaimer

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.

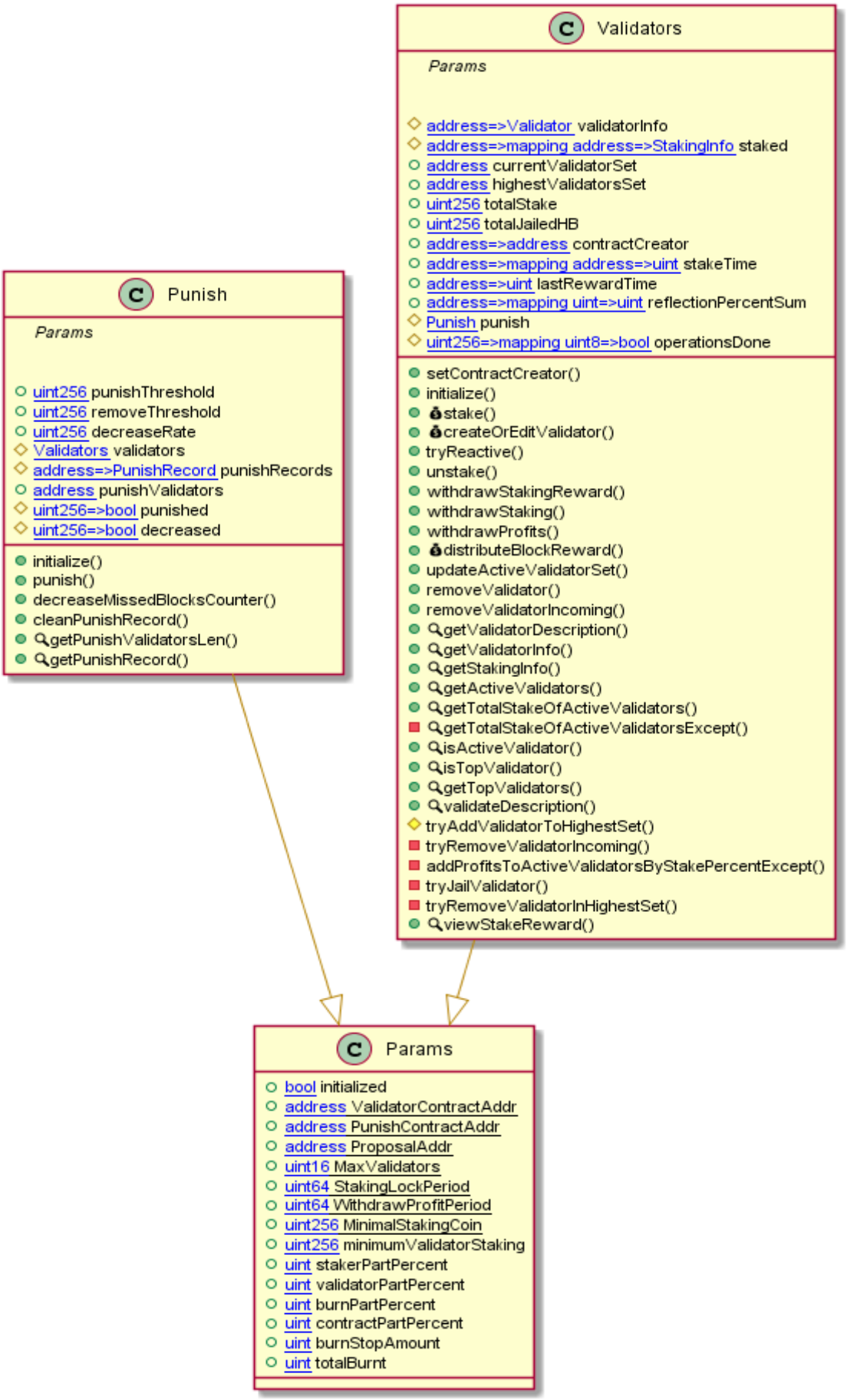
Appendix

Code Flow Diagram - Vention

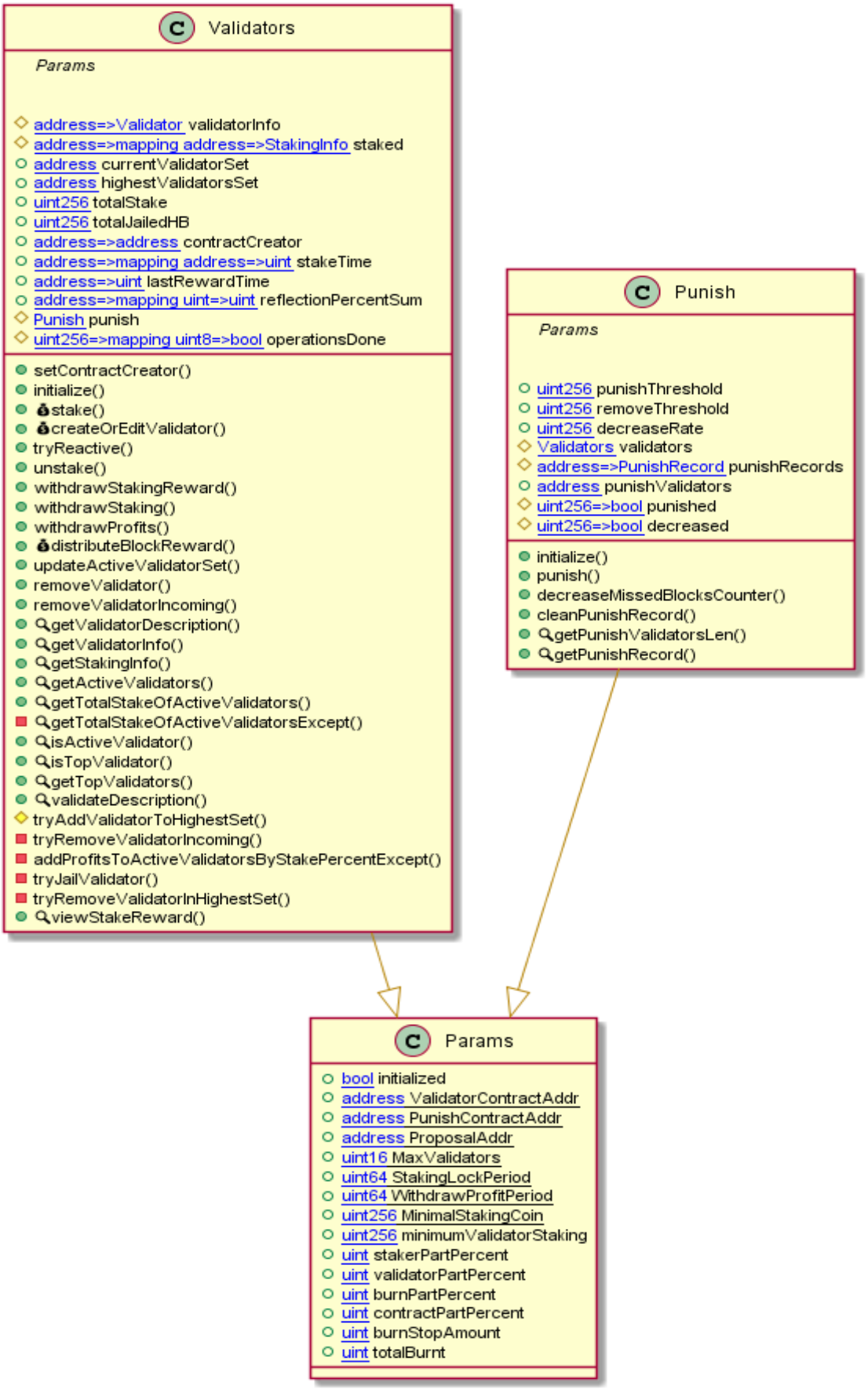
Proposal Diagram



Punish Diagram



Validators Diagram



Slither Results Log

Slither log >> Validators.sol

```
Validators.withdrawStaking(address).staker (Validators.sol#568) lacks a zero-check on :
                - staker.transfer(staking) (Validators.sol#587)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#634-695) has external calls inside a loop: address(contrac
tCreator[_to[i]]).transfer(amt) (Validators.sol#668)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in Validators.tryReactive(address) (Validators.sol#468-490):
        External calls:
        - require(bool,string)(punish.cleanPunishRecord(validator),clean failed) (Validators.sol#483)
        Event emitted after the call(s):
        - LogReactive(validator,block.timestamp) (Validators.sol#487)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Validators.onlyNotRewarded() (Validators.sol#317-323) compares to a boolean constant:
        -require(bool,string)(operationsDone[block.number][uint8(Operations.Distribute)] == false,Block is already rewarded) (V
alidators.sol#318-321)
Validators.onlyNotUpdated() (Validators.sol#325-332) compares to a boolean constant:
        -require(bool,string)(operationsDone[block.number][uint8(Operations.UpdateValidators)] == false,Validators already upda
ted) (Validators.sol#326-330)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Pragma version0.8.4 (Validators.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
```

```
Reentrancy in Validators.withdrawProfits(address) (Validators.sol#594-630):
        External calls:
        - feeAddr.transfer(hbIncoming) (Validators.sol#619)
        Event emitted after the call(s):
        - LogWithdrawProfits(validator,feeAddr,hbIncoming,block.timestamp) (Validators.sol#622-627)
Reentrancy in Validators.withdrawStaking(address) (Validators.sol#567-591):
        External calls:
        - staker.transfer(staking) (Validators.sol#587)
        Event emitted after the call(s):
        - LogWithdrawStaking(staker,validator,staking,block.timestamp) (Validators.sol#589)
Reentrancy in Validators.withdrawStakingReward(address) (Validators.sol#551-565):
        External calls:
        - address(msg.sender).transfer(reward) (Validators.sol#561)
        Event emitted after the call(s):
        - withdrawStakingRewardEv(msg.sender,validator,reward,block.timestamp) (Validators.sol#562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Punish.slitherConstructorVariables() (Validators.sol#74-185) uses literals with too many digits:
        - burnStopAmount = 10000000000000000000000 (Validators.sol#31)
Validators.withdrawStakingReward(address) (Validators.sol#551-565) uses literals with too many digits:
        - reward = stakingInfo.coins * validPercent / 10000000000000000000 (Validators.sol#560)
Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#634-695) uses literals with too many digits:
        - _validatorPart = reward * validatorPartPercent / 100000 (Validators.sol#647)
Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#634-695) uses literals with too many digits:
        - _burnPart = reward * burnPartPercent / 100000 (Validators.sol#651)
Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#634-695) uses literals with too many digits:
        - amt = amt * contractPartPercent / 100000 (Validators.sol#667)
Validators.distributeBlockReward(address[],uint64[]) (Validators.sol#634-695) uses literals with too many digits:
        - reflectionPercentSum[val][lastRewardTime[val]] = lastRewardHold + (remaining * 10000000000000000000 / validatorInfo[
val].coins) (Validators.sol#678)
Validators.viewStakeReward(address,address) (Validators.sol#1037-1047) uses literals with too many digits:
        - stakingInfo.coins * validPercent / 10000000000000000000 (Validators.sol#1043)
Validators.slitherConstructorVariables() (Validators.sol#187-1048) uses literals with too many digits:
```

```
Validators.sol analyzed (3 contracts with 84 detectors), 48 result(s) found
root@server:/chetan/gaza/mycontracts/Aki# slither Params.sol
```

Slither log >> Punish.sol

```
Pragma version0.8.4 (Punish.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Constant Params.ValidatorContractAddr (Punish.sol#9-10) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.PunishContractAddr (Punish.sol#11-12) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.ProposalAddr (Punish.sol#13-14) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.MaxValidators (Punish.sol#17) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.StakingLockPeriod (Punish.sol#19) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.WithdrawProfitPeriod (Punish.sol#21) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.MinimalStakingCoin (Punish.sol#22) is not in UPPER_CASE_WITH_UNDERSCORES
Event ValidatorswithdrawStakingRewardEv(address,address,uint256,uint256) (Punish.sol#316) is not in CapWords
Parameter Validators.setContractCreator(address)._contract (Punish.sol#338) is not in mixedCase
Parameter Validators.distributeBlockReward(address[],uint64[])._to (Punish.sol#635) is not in mixedCase
Parameter Validators.distributeBlockReward(address[],uint64[])._gass (Punish.sol#635) is not in mixedCase
Parameter Validators.viewStakeReward(address,address)._staker (Punish.sol#1038) is not in mixedCase
Parameter Validators.viewStakeReward(address,address)._validator (Punish.sol#1038) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Params.burnPartPercent (Punish.sol#30) should be constant
Params.burnStopAmount (Punish.sol#32) should be constant
Params.contractPartPercent (Punish.sol#31) should be constant
Params.minimumValidatorStaking (Punish.sol#24) should be constant
Params.stakerPartPercent (Punish.sol#28) should be constant
Params.totalBurnt (Punish.sol#33) should be constant
Params.validatorPartPercent (Punish.sol#29) should be constant
Punish.removeThresshold (Punish.sol#77) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
Punish.sol analyzed (3 contracts with 84 detectors), 48 result(s) found
```

Slither log >> Proposal.sol

```
Pragma version0.8.4 (Proposal.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Reentrancy in Validators.tryReactive(address) (Proposal.sol#468-490):
        External calls:
        - require(bool,string)(punish.cleanPunishRecord(validator),clean failed) (Proposal.sol#483)
        Event emitted after the call(s):
        - LogReactive(validator,block.timestamp) (Proposal.sol#487)
Reentrancy in Proposal.voteProposal(bytes32,bool) (Proposal.sol#1153-1208):
        External calls:
        - validators.tryReactive(proposals[id].dst) (Proposal.sol#1193)
        Event emitted after the call(s):
        - LogPassProposal(id,proposals[id].dst,block.timestamp) (Proposal.sol#1194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Proposal.createProposal(address,string) (Proposal.sol#1129-1151) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(proposals[id].createTime == 0,Proposal already exists) (Proposal.sol#1140)
Proposal.voteProposal(bytes32,bool) (Proposal.sol#1153-1208) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(proposals[id].createTime != 0,Proposal not exist) (Proposal.sol#1158)
        - require(bool,string)(block.timestamp < proposals[id].createTime + proposalLastingPeriod,Proposal expired) (Proposal.s
ol#1163-1166)
        - proposals[id].reject >= validators.getActiveValidators().length / 2 + 1 (Proposal.sol#1200-1201)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Pragma version0.8.4 (Proposal.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Punish.slitherConstructorVariables() (Proposal.sol#74-185) uses literals with too many digits:
        - burnStopAmount = 10000000000000000000000000 (Proposal.sol#31)
Validators.withdrawStakingReward(address) (Proposal.sol#551-565) uses literals with too many digits:
        - reward = stakingInfo.coins * validPercent / 10000000000000000000 (Proposal.sol#560)
Validators.distributeBlockReward(address[],uint64[]) (Proposal.sol#634-695) uses literals with too many digits:
        - _validatorPart = reward * validatorPartPercent / 100000 (Proposal.sol#647)
Validators.distributeBlockReward(address[],uint64[]) (Proposal.sol#634-695) uses literals with too many digits:
        - _burnPart = reward * burnPartPercent / 100000 (Proposal.sol#651)
Validators.distributeBlockReward(address[],uint64[]) (Proposal.sol#634-695) uses literals with too many digits:
        - amt = amt * contractPartPercent / 100000 (Proposal.sol#667)
Validators.distributeBlockReward(address[],uint64[]) (Proposal.sol#634-695) uses literals with too many digits:
        - reflectionPercentSum[val][lastRewardTime[val]] = lastRewardHold + (remaining * 10000000000000000000 / validatorInfo[
val].coins) (Proposal.sol#678)
Validators.viewStakeReward(address,address) (Proposal.sol#1037-1047) uses literals with too many digits:
        - stakingInfo.coins * validPercent / 10000000000000000000 (Proposal.sol#1043)
Validators.slitherConstructorVariables() (Proposal.sol#187-1048) uses literals with too many digits:
        - burnStopAmount = 10000000000000000000000000 (Proposal.sol#31)
Proposal.slitherConstructorVariables() (Proposal.sol#1050-1221) uses literals with too many digits:
        - burnStopAmount = 10000000000000000000000000 (Proposal.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Params.burnPartPercent (Proposal.sol#29) should be constant
Params.burnStopAmount (Proposal.sol#31) should be constant
Params.contractPartPercent (Proposal.sol#30) should be constant
Params.minimumValidatorStaking (Proposal.sol#23) should be constant
Params.stakerPartPercent (Proposal.sol#27) should be constant
Params.totalBurnt (Proposal.sol#32) should be constant
Params.validatorPartPercent (Proposal.sol#28) should be constant
Punish.removeThresshold (Proposal.sol#76) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
Proposal.sol analyzed (4 contracts with 84 detectors), 55 result(s) found
```

Email: audit@EtherAuthority.io

Solidity Static Analysis

Proposal.sol

Security

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.
more
Pos: 160:37:

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.
more
Pos: 850:53:

Gas & Economy

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: 74:4:

Gas costs:

Gas requirement of function Punish.cleanPunishRecord 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: 105:4:

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.

more

Pos: 87:8:

Miscellaneous

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.

more

Pos: 678:8:

Punish.sol

Security

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.

more

Pos: 160:37:

Gas costs:

Gas requirement of function Punish.punish 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: 48:4:

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.
more
Pos: 850:53:

Gas & Economy

Gas costs:

Gas requirement of function Punish.punish 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: 48:4:

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:

Miscellaneous

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.
more
Pos: 57:8:

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: 90:16:

Validators.sol

Security

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.

more

Pos: 160:37:

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.

more

Pos: 241:50:

Gas & Economy

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: 164:4:

Gas costs:

Gas requirement of function Validators.isActiveValidator 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: 643:4:

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.
more
Pos: 844:16:

Miscellaneous

Similar variable names:

Validators.viewStakeReward(address,address) : Variables have very similar names "staked" and "_staker". Note: Modifiers are currently not considered by this static analysis.
Pos: 862:52:

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.
more
Pos: 678:8:

Solhint Linter

Proposal.sol

```
Proposal.sol:2:1: Error: Compiler version >=0.6.0 <0.8.0 does not
satisfy the r semver requirement
Proposal.sol:43:5: Error: Explicitly mark visibility of state
Proposal.sol:94:56: Error: Avoid to make time-based decisions in your
business logic
Proposal.sol:128:44: Error: Avoid to make time-based decisions in
your business logic
Proposal.sol:151:57: Error: Avoid to make time-based decisions in
your business logic
Proposal.sol:161:59: Error: Avoid to make time-based decisions in
your business logic
Proposal.sol:175:34: Error: Avoid to make time-based decisions in
your business logic
```

Punish.sol

```
Punish.sol:2:1: Error: Compiler version >=0.6.0 <0.8.0 does not
satisfy the r semver requirement
Punish.sol:23:5: Error: Explicitly mark visibility of state
Punish.sol:24:5: Error: Explicitly mark visibility of state
Punish.sol:72:38: Error: Avoid to make time-based decisions in your
business logic
```

Validators.sol

```
Validators.sol:3:1: Error: Compiler version >=0.6.0 <0.8.0 does not
satisfy the r semver requirement
Validators.sol:10:1: Error: Contract has 19 states declarations but
allowed no more than 15
Validators.sol:92:5: Error: Explicitly mark visibility of
stateValidators.sol:136:9: Error: Variable name must be in mixedCase
Validators.sol:146:5: Error: Event name must be in CamelCase
Validators.sol:171:38: Error: Avoid to use tx.origin
Validators.sol:181:39: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:262:51: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:519:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
Validators.sol:523:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
Validators.sol:535:60: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:566:46: Error: Avoid to make time-based decisions in
```

```
your business logic
Validators.sol:875:54: Error: Avoid to make time-based decisions in
your business logic
```

Software analysis result:

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

Email: audit@EtherAuthority.io