Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Hyperon Chain
Website:	hyperonchain.com
Platform:	Hyperon Chain Network
Language:	Solidity
Date:	November 22nd, 2022

Table of contents

Introduction 4

Project Background 4

Audit Scope 4

Claimed Smart Contract Features 5

Audit Summary 6

Technical Quick Stats 7

Code Quality 8

Documentation 8

Use of Dependencies 8

AS-IS overview 9

Severity Definitions 12

Audit Findings 13

Conclusion 22

Our Methodology 23

Disclaimers 25

Appendix

- Code Flow Diagram 26
- Slither Results Log 29
- Solidity static analysis 33
- Solhint Linter 44

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 Hyperon Chain to perform the Security audit of the Hyperon Chain 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 November 22nd, 2022.

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 audit scope consists of system smart contracts of the hyperon chain. 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 Hyperon Chain System Smart Contracts
Platform	**Hyperon Chain Network / Solidity**
File 1	Validators.sol
File 2	Staking.sol
File 3	Punish.sol
File 4	WHPN.sol
File 5	Params.sol
Audit Date	November 22nd, 2022
Revised Audit Date	November 29th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
HPN Tokenomics • Coin name: HyperonChain • Coin symbol: HPN • Decimal: 18 • Total Supply: 120 Million • No more coins generated ever	**YES, This is valid.**
File 1 Validators.sol • Maximum Validators: 21 • Staking Lock Period: 10 days • Minimal Staking Coin: 32 HPN • Maximum Reward: 12 Million HPN • Block reward generated from transaction fees. • Validators will receive 15% of the block reward	**YES, This is valid.**
File 2 Staking.sol • 85% of block reward goes to staking smart contracts, where it will be distributed to master voters, staking voters and no-staking voters. • This contract can be changed anytime by the owner until the ownership is renounced.	**YES, This is valid.**
File 3 Punish.sol • The validator can be punished for misbehavior. • Validators can clean validator's punish records if one restake in.	**YES, This is valid.**
File 4 WHPN.sol • A wrapped token on HPN which exchange value 1:1	**YES, This is valid.**
File 5 Params.sol • It holds parameters of other smart contracts	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, 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, 1 high, 0 medium and 1 low and some very low level issues.
All the issues have been fixed / acknowledged in the revised code.

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	Passed
	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	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	Moderated
	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**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 5 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Hyperon Chain 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 Hyperon Chain Protocol.

The Hyperon Chain team has not provided unit test scripts, which would not help to determine the integrity of the code in an automated way.

All code parts are not well commented on smart contracts.

Documentation

We were given a Hyperon Chain smart contract code in the form of a Github link. 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 website: https://hyperonchain.com which provided rich information about the project architecture.

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

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	initialize	external	Infinite loops possibility	No Issue
5	stake	external	Passed	No Issue
6	createOrEditValidator	external	access only Initialized	No Issue
7	tryReactive	external	access only Initialized	No Issue
8	unstake	external	Passed	No Issue
9	withdrawStaking	external	Passed	No Issue
10	withdrawableReward	read	Passed	No Issue
11	withdrawProfits	external	Passed	No Issue
12	distributeBlockReward	external	Infinite loops possibility	No Issue
13	updateActiveValidatorSet	write	access only Miner	No Issue
14	removeValidator	external	access only Punish Contract	No Issue
15	removeValidatorIncoming	external	access only Punish Contract	No Issue
16	getActiveValidators	read	Passed	No Issue
17	getTotalStakeOfActiveValidators	read	Passed	No Issue
18	getTotalStakeOfActiveValidators Except	read	Passed	No Issue
19	getTotalStakeOfHighestValidator sExcept	read	Passed	No Issue
20	tryAddValidatorToHighestSet	internal	Passed	No Issue
21	tryRemoveValidatorIncoming	write	Passed	No Issue
22	addProfitsToActiveValidatorsBySt akePercentExcept	write	Passed	No Issue
23	tryJailValidator	write	Passed	No Issue
24	tryRemoveValidatorInHighestSet	write	Passed	No Issue
25	getValidatorInfo	read	Passed	No Issue
26	getStakingInfo	read	Passed	No Issue
27	getActiveValidators	read	Passed	No Issue
28	getTotalStakeOfActiveValidators	read	Passed	No Issue
29	isActiveValidator	read	Passed	No Issue
30	isTopValidator	read	Passed	No Issue
31	getTopValidators	read	Passed	No Issue
32	validateDescription	read	Passed	No Issue

33	getTotalStakeOfHighestValidatorsExcept	read	Passed	No Issue
34	emrgencyWithdrawFund	write	access onlyOwner	No Issue
35	onlyMiner	modifier	Passed	No Issue
36	onlyNotInitialized	modifier	Passed	No Issue
37	onlyInitialized	modifier	Passed	No Issue
38	onlyPunishContract	modifier	Passed	No Issue
39	onlyBlockEpoch	modifier	Passed	No Issue
40	onlyValidatorsContract	modifier	Passed	No Issue
41	onlyProposalContract	modifier	Passed	No Issue

Punish.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
9	onlyNotPunished	modifier	Passed	No Issue
10	onlyNotDecreased	modifier	Passed	No Issue
11	initialize	external	access only NotInitialized	No Issue
12	punish	external	access only Miner	No Issue
13	decreaseMissedBlocksCounter	external	access only Miner	No Issue
14	cleanPunishRecord	external	access only Initialized	No Issue
15	getPunishValidatorsLen	read	Passed	No Issue
16	getPunishRecord	read	Passed	No Issue

Staking.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	setWHPN	write	called only once	No Issue
2	setValidator	write	access onlyowner	No Issue
3	initialize	write	called only once	No Issue
4	stake	write	Passed	No Issue
5	stakeForMaster	write	Passed	No Issue
6	unstake	write	Passed	No Issue
7	withdrawStakingReward	write	Passed	No Issue
8	withdrawEmergency	write	Passed	No Issue
9	withdrawStaking	write	Passed	No Issue
10	withdrawableReward	write	Passed	No Issue
11	calculateReflectionPercent	read	Passed	No Issue
12	setValidators	write	high gas consuming loop	No Issue
13	addProfitsToActiveValidatorsByStakePercentExcept	write	high gas consuming loop	No Issue
14	dividendsOf	read	Passed	No Issue
15	getValidatorInfo	read	Passed	No Issue
16	getMasterVoterInfo	read	Passed	No Issue
17	getTotalStakeOfHighestValidatorsExcept	read	Passed	No Issue
18	emrgencyWithdrawFund	write	access only owner	No Issue

WHPN.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	mint	write	Tokens minted as per incoming HPN	No Issue
2	burn	write	Tokens burned as per outgoing HPN	No Issue
3	_beforeTokenTransfer	internal	Sets the variable of last token transfer	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 loss
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

(1) Invalid address pass: **- Validators.sol**

```
function withdrawableReward(address validator, address _user) public view returns(uint)
{

    StakingInfo memory stakingInfo = staked[_user][validator];
    uint256 validPercent ;
    uint256 _lastTransferTime =  WHPN.lastTransfer(msg.sender);
    if(_lastTransferTime < staked[msg.sender][validator].stakeTime)
    {
      validPercent = reflectionPercentSum[validator][lastRewardTime[validator]] - reflection
    }
    if(masterVoterInfo[_user].coins>0)
    {
      validPercent = reflectionMasterPerent[validator][lastRewardTime[validator]] - reflecti
    }
    if(stakingInfo.coins == 0)
```

There is a function withdrawableReward(), to get _lastTransferTime. But the passed wallet address is invalid. is should be _user instead of msg.sender.

Resolution: Change msg.sender to _user for get _lastTransferTime.

Status: This issue is fixed in the revised contract code

Medium

No medium severity vulnerabilities were found.

Low

No low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Same functionality found: **- Validators.sol**

```
uint256 unstakeAmount = stakingInfo.coins;
require(unstakeAmount > 0, "You don't have any stake");
// You can't unstake if the validator is the only one top validator and
```

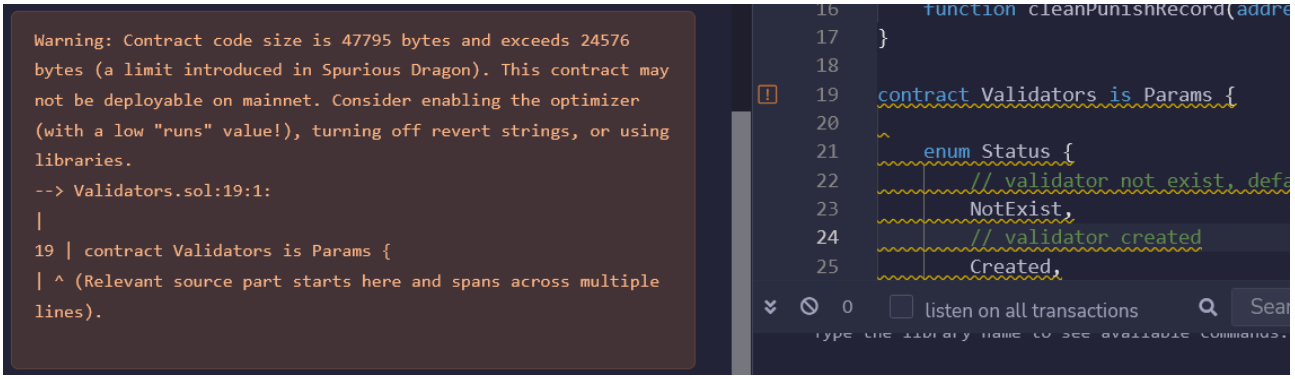
```
stakingInfo.index = 0;
stakingInfo.coins = 0;
require(unstakeAmount > 0, "You don't have any stake");
payable(staker).transfer(unstakeAmount);
```

There is an "unstake()" function that has the same required "unstakeAmount > 0" condition checked in code.

Resolution: We suggest removing duplicate required conditions from code.

Status: This issue is fixed in the revised contract code

(2) Validators contract code size limit: **- Validators.sol**



Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off revert strings, or using libraries.

Resolution: We suggest removing unnecessary code blocks and optimizing the code.

Status: This issue is fixed in the revised contract code

(3) Unused event: **- Validators.sol**

LogWithdrawStaking() event is defined, but not used in code.

Resolution: We suggest removing unused events.

Status: This issue is fixed in the revised contract code

(4) Deprecated function use: **- Validators.sol**

```
function initialize(address[] calldata vals) external onlyNotInitialized {
    proposal = Proposal(ProposalAddr);
    punish = Punish(PunishContractAddr);
    signer = 0xBFb9B248D0e735032a70826572f79381dDC7F0De;

    for (uint256 i = 0; i < vals.length; i++) {
        require(vals[i] != address(0), "Invalid validator address");

        lastRewardTime[vals[i]] = now;
        reflectionPercentSum[vals[i]][lastRewardTime[vals[i]]] = 0;
        reflectionMasterPerent[vals[i]][lastRewardTime[vals[i]]] = 0;
```

In the initialize function, “now” has been deprecated. Use "block.timestamp" instead.

Resolution: We suggest using "block.timestamp" instead of "now".

Status: This issue is fixed in the revised contract code

(5) Function input parameters lack of check: **- Validators.sol**

```
function stake(address validator)
    external
    payable
    onlyInitialized
    returns (bool)
{
    address payable staker = msg.sender;
    uint256 staking = msg.value;

    require(
        validatorInfo[validator].status == Status.Created ||
            validatorInfo[validator].status == Status.Staked,
        "Can't stake to a validator in abnormal status"
    );
    require(
        proposal.pass(validator),
        "The validator you want to stake must be authorized first"
    );
    //****************************
    bool isMaster;
    if(staking >= totalsupply.mul(2).div(100))
    {
      isMaster = true;
    }
```

Variable validation is not performed in below functions:

- stake
- unstake
- withdrawProfits

Resolution: We advise to put validation: address type variables should not be address(0).

Status: This issue is fixed in the revised contract code

(6) Spelling mistake: **- Validators.sol**

```
MasterVoter storage masterInfo = masterVoterInfo[_masterVoter];
// stake at first time to this valiadtor
if (stakedMaster[staker][_masterVoter].coins == 0) {
```

Spelling mistakes in comments.

“valiadtor” word should be **“validator.”**

Resolution: Correct the spelling.

Status: This issue is fixed in the revised contract code

(7) Critical operation lacks event log: **- Validators.sol**

```
function emrgencyWithdrawFund()external {
    require(msg.sender==Proxy(ValidatorContractAddr).owner());
    payable(msg.sender).transfer(address(this).balance);
}
```

Missing event log for:

- initialize()
- emrgencyWithdrawFund()

Resolution: Please write an event log for listed events.

Status: This issue is acknowledged in the revised contract code

(8) Infinite loops possibility:

Validators.sol

```
// this is initialized by the blockchain itself.
// so no need to initialize separately.
function initialize(address[] calldata vals) external onlyNotInitialized {

    punish = IPunish(PunishContractAddr);

    for (uint256 i = 0; i < vals.length; i++) {
        require(vals[i] != address(0), "err1");
```

Validators.sol

```
function initialize(address[] calldata vals) external onlyNotInitialized {
    proposalLastingPeriod = 7 days;
    validators = IValidator(ValidatorContractAddr);

    for (uint256 i = 0; i < vals.length; i++) {
        require(vals[i] != address(0), "Invalid validator address");
        pass[vals[i]] = true;
    }
```

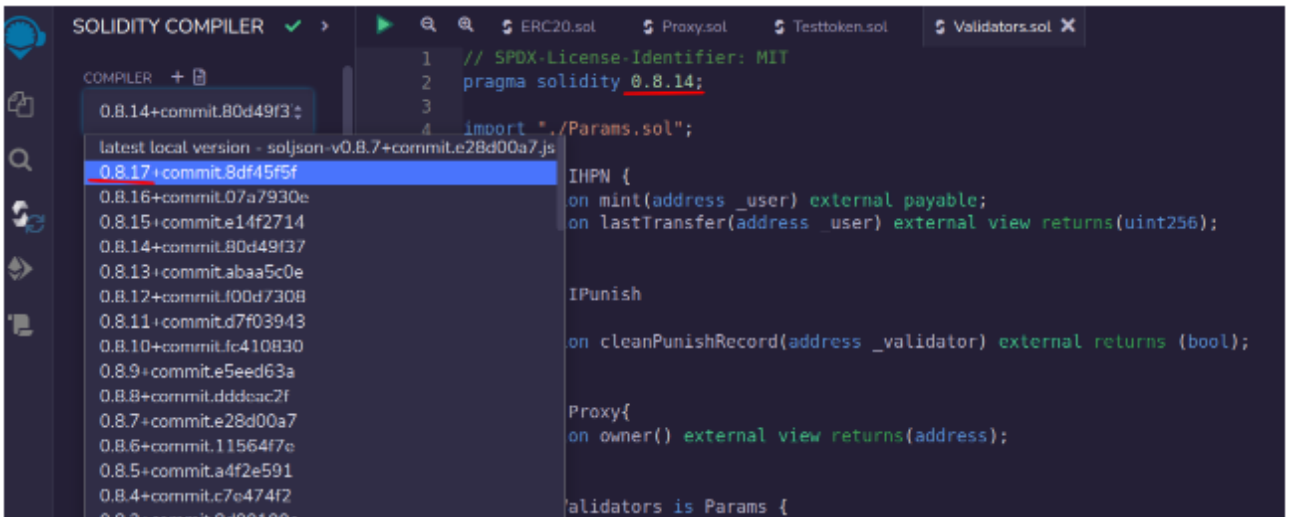
As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

Resolution: Adjust logic to replace loops with mapping or other code structure.

- initialize() - vals.length.

Status: This issue is acknowledged in the revised contract code

(9) Please use the latest compiler version when deploying contract: **- Validators.sol and Punish.sol**



This is not a severe issue, but we suggest using the latest compiler version at the time of contract deployment, which is 0.8.17 at the time of this audit. Using the latest compiler version is always recommended which prevents any compiler level issues.

Status: This issue is acknowledged in the revised contract code

(10) Unlocked Compiler Version: **- Validators.sol, Punish.sol**

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.14;
```

The contract uses the "^" prefix specifier, Use the Unlocked compiler version. Unlocked compiler version code of the smart contract, and that gives permission to the users to compile it one higher than a particular version.

Resolution: We suggest using that the compiler version is unlocked instead of the locked compiler version. The following line of code can be added to the project:
pragma solidity 0.8.14;

Status: This issue is fixed in the revised contract code

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:

- setStakingContract function in Validators contract. This sets a staking contract.
- emrgencyWithdrawFund function in Validators contract. This allows the owner to withdraw all the funds from the validator smart contract.
- setValidator function in Staking contract. This sets the validator address in the staking contract.
- emrgencyWithdrawFund function in Staking contract. This allows the owner to withdraw all the funds from the staking smart contract.

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 link. And we have used all possible tests based on given objects as files. We have observed 1 high severity issue, 1 low severity issue and some informational issues in the smart contracts. All the issues have been fixed / acknowledged in the revised code. So, **it's good to go for the mainnet deployment**.

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 **"Secure".**

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.

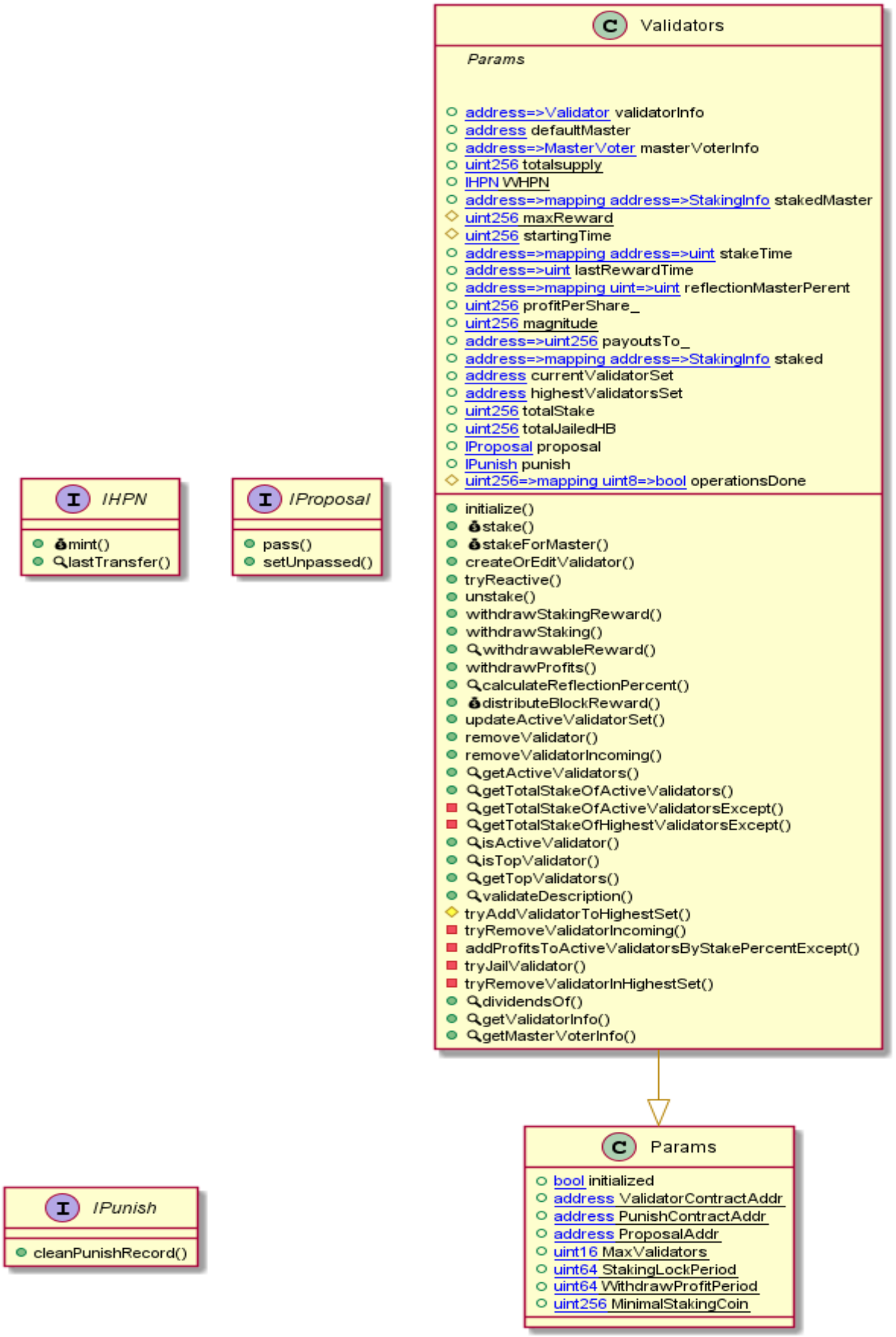
Technical Disclaimer

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.

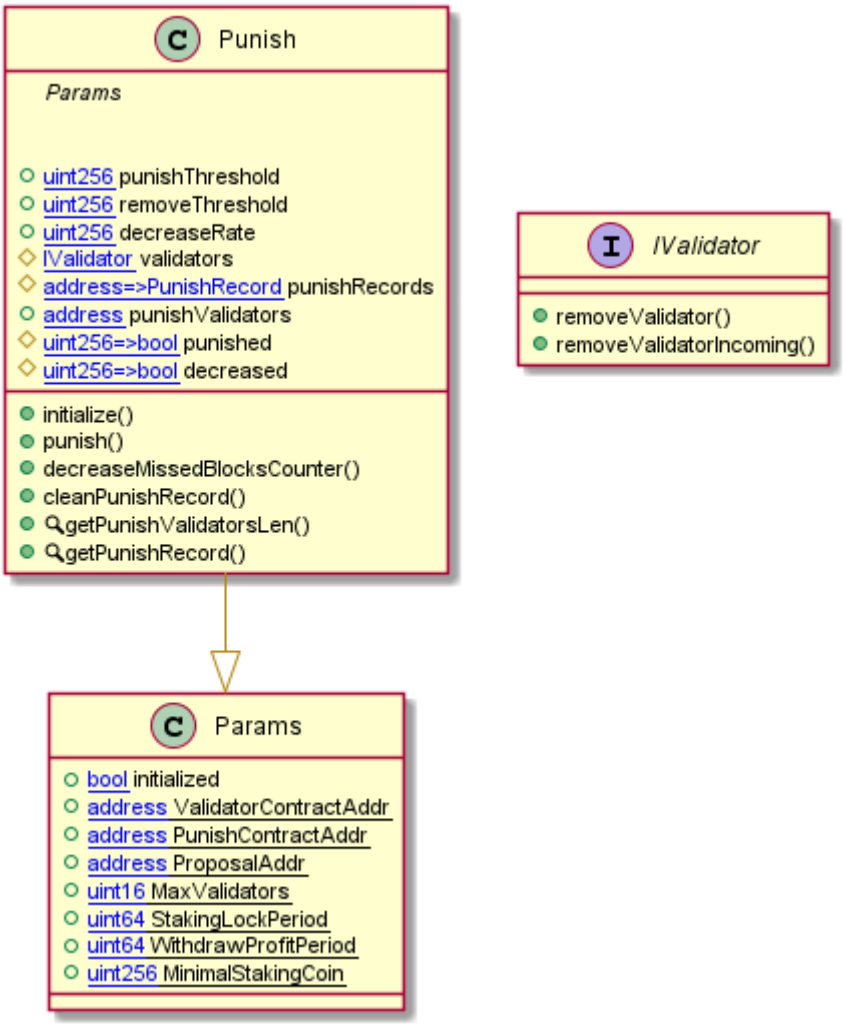
Appendix

Code Flow Diagram - Hyperon Chain Protocol

Validators Diagram



Punish Diagram



Slither Results Log

Slither Log >> Validators.sol

```
INFO:Detectors:
Reentrancy in Validators.stake(address) (Validators.sol#284-369):
        External calls:
        - require(bool,string)(proposal.pass(validator),The validator you want to stake must be authorized first) (Validators.s
ol#298-301)
        State variables written after the call(s):
        - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#337)
                - highestValidatorsSet.push(val) (Validators.sol#919)
                - highestValidatorsSet[lowestIndex] = val (Validators.sol#945)
        - masterInfo.coins += staking (Validators.sol#348)
        - masterInfo.validator = validator (Validators.sol#352)
        - masterInfo.stakers.push(staker) (Validators.sol#354)
        - payoutsTo_[staker] += dividendsOf(staker,validator) * magnitude (Validators.sol#330)
        - payoutsTo_[staker] += profitPerShare_ * 3 * staking (Validators.sol#358)
        - stakeTime[staker][validator] = lastRewardTime[validator] (Validators.sol#326)
        - staked[staker][validator].index = valInfo.stakers.length (Validators.sol#324)
        - staked[staker][validator].coins += staking (Validators.sol#340)
        - staked[staker][validator].stakeTime = block.timestamp (Validators.sol#342)
        - stakedMaster[staker][staker].index = masterInfo.stakers.length (Validators.sol#353)
        - stakedMaster[staker][staker].coins += staking (Validators.sol#356)
        - totalStake += staking (Validators.sol#343)
Reentrancy in Validators.stakeForMaster(address) (Validators.sol#371-428):
        External calls:
        - require(bool,string)(proposal.pass(validator),10) (Validators.sol#386-389)
        State variables written after the call(s):
        - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#420)
                - highestValidatorsSet.push(val) (Validators.sol#919)
                - highestValidatorsSet[lowestIndex] = val (Validators.sol#945)
        - payoutsTo_[staker] += dividendsOf(staker,_masterVoter) * magnitude (Validators.sol#411)
        - payoutsTo_[staker] += profitPerShare_ * 3 * staking (Validators.sol#414)
        - stakeTime[staker][_masterVoter] = lastRewardTime[validator] (Validators.sol#407)
        - staked[_masterVoter][validator].coins += staking (Validators.sol#423)
        - stakedMaster[staker][_masterVoter].index = masterInfo.stakers.length (Validators.sol#405)
        - stakedMaster[staker][_masterVoter].coins += staking (Validators.sol#413)
        - totalStake += staking (Validators.sol#424)
Reentrancy in Validators.unstake(address) (Validators.sol#494-620):
```

```
        External calls:
        - proposal.setUnpassed(validator) (Validators.sol#612)
        External calls sending eth:
        - withdrawStakingReward(validator) (Validators.sol#615)
                - staker.transfer(reward) (Validators.sol#649)
        State variables written after the call(s):
        - withdrawStakingReward(validator) (Validators.sol#615)
                - payoutsTo_[staker] += profitPerShare_ * staked[staker][validatorOrMastervoter].coins (Validators.sol#643)
        - withdrawStakingReward(validator) (Validators.sol#615)
                - stakeTime[staker][validatorOrMastervoter] = lastRewardTime[validatorOrMastervoter] (Validators.sol#648)
        - stakeTime[staker][validator] = 0 (Validators.sol#616)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Validators.createOrEditValidator(address,string,string,string,string,string) (Validators.sol#430-469):
        External calls:
        - require(bool,string)(proposal.pass(validator),You must be authorized first) (Validators.sol#446)
        Event emitted after the call(s):
        - LogCreateValidator(validator,feeAddr,block.timestamp) (Validators.sol#464)
        - LogEditValidator(validator,feeAddr,block.timestamp) (Validators.sol#466)
Reentrancy in Validators.removeValidator(address) (Validators.sol#799-814):
        External calls:
        - proposal.setUnpassed(val) (Validators.sol#811)
        Event emitted after the call(s):
        - LogRemoveValidator(val,hb,block.timestamp) (Validators.sol#812)
Reentrancy in Validators.stake(address) (Validators.sol#284-369):
        External calls:
        - require(bool,string)(proposal.pass(validator),The validator you want to stake must be authorized first) (Validators.s
ol#298-301)
        Event emitted after the call(s):
        - LogAddToTopValidators(val,block.timestamp) (Validators.sol#920)
                - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#337)
        - LogAddToTopValidators(val,block.timestamp) (Validators.sol#940)
                - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#337)
        - LogRemoveFromTopValidators(highestValidatorsSet[lowestIndex],block.timestamp) (Validators.sol#941-944)
                - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#337)
Reentrancy in Validators.stake(address) (Validators.sol#284-369):
```

```
        External calls:
        - require(bool,string)(proposal.pass(validator),The validator you want to stake must be authorized first) (Validators.s
ol#298-301)
        - WHPN.mint{value: staking}(staker) (Validators.sol#363)
        External calls sending eth:
        - WHPN.mint{value: staking}(staker) (Validators.sol#363)
        Event emitted after the call(s):
        - LogStake(staker,validator,staking,block.timestamp) (Validators.sol#367)
Reentrancy in Validators.stakeForMaster(address) (Validators.sol#371-428):
        External calls:
        - require(bool,string)(proposal.pass(validator),10) (Validators.sol#386-389)
        Event emitted after the call(s):
        - LogAddToTopValidators(val,block.timestamp) (Validators.sol#920)
                - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#420)
        - LogAddToTopValidators(val,block.timestamp) (Validators.sol#940)
                - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#420)
        - LogRemoveFromTopValidators(highestValidatorsSet[lowestIndex],block.timestamp) (Validators.sol#941-944)
                - tryAddValidatorToHighestSet(validator,valInfo.coins) (Validators.sol#420)
        - LogStakeForMaster(staker,_masterVoter,staking,block.timestamp) (Validators.sol#426)
```

```
Reentrancy in Validators.tryReactive(address) (Validators.sol#471-492):
        External calls:
        - require(bool,string)(punish.cleanPunishRecord(validator),clean failed) (Validators.sol#486)
        Event emitted after the call(s):
        - LogReactive(validator,block.timestamp) (Validators.sol#490)
Reentrancy in Validators.unstake(address) (Validators.sol#494-620):
        External calls:
        - proposal.setUnpassed(validator) (Validators.sol#612)
        External calls sending eth:
        - withdrawStakingReward(validator) (Validators.sol#615)
                - staker.transfer(reward) (Validators.sol#649)
        Event emitted after the call(s):
        - LogUnstake(staker,validator,unstakeAmount,block.timestamp) (Validators.sol#618)
        - withdrawStakingRewardEv(staker,validatorOrMastervoter,reward,block.timestamp) (Validators.sol#650)
                - withdrawStakingReward(validator) (Validators.sol#615)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Validators.withdrawStakingReward(address) (Validators.sol#622-652) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stakeTime[staker][validatorOrMastervoter] < lastRewardTime[validatorOrMastervoter],no reward yet
) (Validators.sol#635)
Validators.distributeBlockReward() (Validators.sol#753-782) uses timestamp for comparisons
        Dangerous comparisons:
        - modDuration != rInfo.rewardDuration (Validators.sol#770)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Validators.onlyNotRewarded() (Validators.sol#237-243) compares to a boolean constant:
        -require(bool,string)(operationsDone[block.number][uint8(Operations.Distribute)] == false,Block is already rewarded) (V
alidators.sol#238-241)
Validators.onlyNotUpdated() (Validators.sol#245-252) compares to a boolean constant:
        -require(bool,string)(operationsDone[block.number][uint8(Operations.UpdateValidators)] == false,Validators already upda
ted) (Validators.sol#246-250)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
INFO:Detectors:
Validators.addProfitsToActiveValidatorsByStakePercentExcept(uint256,address) (Validators.sol#973-1049) has costly operations in
side a loop:
        - profitPerShare_ += PerCoin_Reward * magnitude (Validators.sol#1041)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Pragma version0.8.4 (Validators.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Constant Params.ValidatorContractAddr (Validators.sol#8-9) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.PunishContractAddr (Validators.sol#10-11) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.ProposalAddr (Validators.sol#12-13) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.MaxValidators (Validators.sol#16) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.StakingLockPeriod (Validators.sol#18) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.WithdrawProfitPeriod (Validators.sol#20) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.MinimalStakingCoin (Validators.sol#21) is not in UPPER_CASE_WITH_UNDERSCORES
Struct Validators.rewardInfo (Validators.sol#139-144) is not in CapWords
Event ValidatorswithdrawStakingRewardEv(address,address,uint256,uint256) (Validators.sol#235) is not in CapWords
Parameter Validators.stakeForMaster(address)._masterVoter (Validators.sol#371) is not in mixedCase
Parameter Validators.withdrawableReward(address,address)._user (Validators.sol#680) is not in mixedCase
Parameter Validators.calculateReflectionPercent(uint256,uint256)._totalAmount (Validators.sol#748) is not in mixedCase
Parameter Validators.calculateReflectionPercent(uint256,uint256)._rewardAmount (Validators.sol#748) is not in mixedCase
Parameter Validators.dividendsOf(address,address)._user (Validators.sol#1085) is not in mixedCase
Constant Validators.totalsupply (Validators.sol#134) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Validators.maxReward (Validators.sol#138) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Validators.magnitude (Validators.sol#154) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in Validators.unstake(address) (Validators.sol#494-620):
        External calls:
        - withdrawStakingReward(validator) (Validators.sol#615)
                - staker.transfer(reward) (Validators.sol#649)
        State variables written after the call(s):
        - stakeTime[staker][validator] = 0 (Validators.sol#616)
        Event emitted after the call(s):
        - LogUnstake(staker,validator,unstakeAmount,block.timestamp) (Validators.sol#618)
```

```
Reentrancy in Validators.withdrawProfits(address) (Validators.sol#710-746):
        External calls:
        - feeAddr.transfer(hbIncoming) (Validators.sol#735)
        Event emitted after the call(s):
        - LogWithdrawProfits(validator,feeAddr,hbIncoming,block.timestamp) (Validators.sol#738-743)
Reentrancy in Validators.withdrawStaking(address) (Validators.sol#654-678):
        External calls:
        - staker.transfer(staking) (Validators.sol#674)
        Event emitted after the call(s):
        - LogWithdrawStaking(staker,validator,staking,block.timestamp) (Validators.sol#676)
Reentrancy in Validators.withdrawStakingReward(address) (Validators.sol#622-652):
        External calls:
        - staker.transfer(reward) (Validators.sol#649)
        Event emitted after the call(s):
        - withdrawStakingRewardEv(staker,validatorOrMastervoter,reward,block.timestamp) (Validators.sol#650)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Validators.calculateReflectionPercent(uint256,uint256) (Validators.sol#748-750) uses literals with too many digits:
        - (_rewardAmount * 10000000000000000000 / _totalAmount) / (1000000000000000000) (Validators.sol#749)
Validators.slitherConstructorConstantVariables() (Validators.sol#76-1119) uses literals with too many digits:
        - ValidatorContractAddr = 0x000000000000000000000000000000000000f000 (Validators.sol#8-9)
Validators.slitherConstructorConstantVariables() (Validators.sol#76-1119) uses literals with too many digits:
        - PunishContractAddr = 0x000000000000000000000000000000000000F001 (Validators.sol#10-11)
Validators.slitherConstructorConstantVariables() (Validators.sol#76-1119) uses literals with too many digits:
        - ProposalAddr = 0x000000000000000000000000000000000000F002 (Validators.sol#12-13)
Validators.slitherConstructorConstantVariables() (Validators.sol#76-1119) uses literals with too many digits:
        - totalsupply = 100000000000000000 (Validators.sol#134)
Validators.slitherConstructorConstantVariables() (Validators.sol#76-1119) uses literals with too many digits:
        - maxReward = 12000000 * 1e18 (Validators.sol#138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

Slither Log >> Punish.sol

```
INFO:Detectors:
Params.onlyBlockEpoch(uint256) (Punish.sol#43-46) uses a dangerous strict equality:
        - require(bool,string)(block.number % epoch == 0,Block epoch only) (Punish.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Punish.punish(address) (Punish.sol#110-135):
        External calls:
        - validators.removeValidator(val) (Punish.sol#125)
        State variables written after the call(s):
        - punishRecords[val].missedBlocksCounter = 0 (Punish.sol#127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Reentrancy in Punish.punish(address) (Punish.sol#110-135):
        External calls:
        - validators.removeValidator(val) (Punish.sol#125)
        - validators.removeValidatorIncoming(val) (Punish.sol#131)
        Event emitted after the call(s):
        - LogPunishValidator(val,block.timestamp) (Punish.sol#134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Pragma version0.8.4 (Punish.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Constant Params.ValidatorContractAddr (Punish.sol#8-9) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.PunishContractAddr (Punish.sol#10-11) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.ProposalAddr (Punish.sol#12-13) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.MaxValidators (Punish.sol#16) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.StakingLockPeriod (Punish.sol#18) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.WithdrawProfitPeriod (Punish.sol#20) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Params.MinimalStakingCoin (Punish.sol#21) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
Punish.slitherConstructorConstantVariables() (Punish.sol#69-200) uses literals with too many digits:
        - ValidatorContractAddr = 0x0000000000000000000000000000000000001000 (Punish.sol#8-9)
Punish.slitherConstructorConstantVariables() (Punish.sol#69-200) uses literals with too many digits:
        - PunishContractAddr = 0x0000000000000000000000000000000000001001 (Punish.sol#10-11)
Punish.slitherConstructorConstantVariables() (Punish.sol#69-200) uses literals with too many digits:
        - ProposalAddr = 0x0000000000000000000000000000000000001002 (Punish.sol#12-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
getPunishValidatorsLen() should be declared external:
        - Punish.getPunishValidatorsLen() (Punish.sol#193-195)
getPunishRecord(address) should be declared external:
        - Punish.getPunishRecord(address) (Punish.sol#197-199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Punish.sol analyzed (3 contracts with 75 detectors), 17 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Validators.sol

Security

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: 977:38:

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: 1022:53:

Gas & Economy

Gas costs:

Gas requirement of function Validators.stake 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: 227:4:

Gas costs:

Gas requirement of function Validators.stakeForMaster 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: 314:4:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function Validators.removeValidator 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: 742:4:

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: 197:4:

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: 811:4:

Gas costs:

Gas requirement of function Validators.isTopValidator 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: 821:4:

Gas costs:

Gas requirement of function Validators.removeValidator 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: 742:4:

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: 943:12:

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: 1016:16:

Miscellaneous

Similar variable names:

Validators.withdrawStaking(address) : Variables have very similar names "staked" and "staker". Note: Modifiers are currently not considered by this static analysis.

Pos: 617:8:

Similar variable names:

Validators.withdrawStaking(address) : Variables have very similar names "staked" and "staker". Note: Modifiers are currently not considered by this static analysis.

Pos: 619:32:

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: 121:8:

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: 122:12:

Miscellaneous

Constant/View/Pure functions:

IValidator.removeValidator(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 7:4:

Similar variable names:

Punish.cleanPunishRecord(address) : Variables have very similar names "uval" and "val". Note: Modifiers are currently not considered by this static analysis.

Pos: 130:26:

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: 40:8:

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: 99:20:

Solhint Linter

Validators.sol

```
Validators.sol:2:1: Error: Compiler version 0.8.14 does not satisfy
the r semver requirement
Validators.sol:19:1: Error: Contract has 18 states declarations but
allowed no more than 15
Validators.sol:77:29: Error: Constant name must be in capitalized
SNAKE_CASE
Validators.sol:81:5: Error: Explicitly mark visibility of state
Validators.sol:81:22: Error: Constant name must be in capitalized
SNAKE_CASE
Validators.sol:82:5: Error: Contract name must be in CamelCase
Validators.sol:88:5: Error: Explicitly mark visibility of state
Validators.sol:97:29: Error: Constant name must be in capitalized
SNAKE_CASE
Validators.sol:119:5: Error: Explicitly mark visibility of state
Validators.sol:178:5: Error: Event name must be in CamelCase
Validators.sol:204:39: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:223:24: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:285:47: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:310:51: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:369:63: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:407:57: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:409:55: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:433:37: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:561:59: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:593:77: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:619:61: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:685:13: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:712:35: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:724:52: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:755:46: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:863:45: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:883:41: Error: Avoid to make time-based decisions in
your business logic
```

```
Validators.sol:886:13: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:912:50: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:977:39: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:983:17: Error: Variable name must be in mixedCase
Validators.sol:1022:54: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:1030:9: Error: Visibility modifier must be first in
list of modifiers
```

Punish.sol

```
Punish.sol:2:1: Error: Compiler version 0.8.12 does not satisfy the r
semver requirement
Punish.sol:23:5: Error: Explicitly mark visibility of state
Punish.sol:25:5: Error: Explicitly mark visibility of state
Punish.sol:28:5: Error: Explicitly mark visibility of state
Punish.sol:29:5: Error: Explicitly mark visibility of state
Punish.sol:77:38: Error: Avoid to make time-based decisions in your
business logic
```

Staking.sol

```
Staking.sol:2:1: Error: Compiler version 0.8.12 does not satisfy the
r semver requirement
Staking.sol:48:5: Error: Explicitly mark visibility of state
Staking.sol:99:56: Error: Avoid to make time-based decisions in your
business logic
Staking.sol:108:31: Error: Avoid to make time-based decisions in your
business logicStaking.sol:133:44: Error: Avoid to make time-based
decisions in your business logic
Staking.sol:156:57: Error: Avoid to make time-based decisions in your
business logic
Staking.sol:166:59: Error: Avoid to make time-based decisions in your
business logic
Staking.sol:180:34: 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