

SMART CONTRACT

Security Audit Report

Project: MainnetZ Chain
Website: <https://mainnetz.io>
Platform: MainnetZ Chain
Language: Solidity
Date: January 3rd, 2023

Table of contents

Introduction	4
Project Background	4
Audit Scope	5
Claimed Smart Contract Features	6
Audit Summary	8
Technical Quick Stats	9
Code Quality	10
Documentation	10
Use of Dependencies	10
AS-IS overview	11
Severity Definitions	15
Audit Findings	16
Conclusion	21
Our Methodology	22
Disclaimers	24
Appendix	
• Code Flow Diagram	25
• Slither Results Log	30
• Solidity static analysis	34
• Solhint Linter	39

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.

Introduction

EtherAuthority was contracted by MainnetZ Chain to perform the Security audit of the MainnetZ Chain 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 January 3rd, 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

- MainnetZ empowers the use and creation of decentralized applications (Dapps) and to not only be the leading edge communal blockchain, but to also focus on the revelations of developers releasing innovative projects with high potential.
- MainnetZ strives to become the origin of innovative Dapp technology, promotion, business, and manufacturing within the cryptocurrency sector.
- The audit scope consists of system smart contracts of the MainnetZ Chain. The system smart contracts contribute heavily to the consensus mechanism.
- The system smart contracts performs actions such as Create/Update Validators, Vote for Proposals, staking for validators, Bridge, etc.

Audit scope

Name	Code Review and Security Analysis Report for MainnetZ Chain System Smart Contracts
Platform	MainnetZ Chain / Solidity
File 1	Bridge.sol
File 1 Github Commit	F08C4ABC74B1089F079FB2AD695EEA3A
File 2	PeggedToken.sol
File 2 Github Commit	045096375994504E5DD9296960D51ED7
File 3	Proposal.sol
File 3 Github Commit	D41AF773FE41436CBCBF4F112DE6385D
File 4	Punish.sol
File 4 Github Commit	69BAE7AF30D60741D97B0F5C53B43A46
File 5	Validators.sol
File 5 Github Commit	ACB6B049843D84049B3A0E7B9086BD9C
Audit Date	January 3rd, 2023
Revised Audit Date	January 13th, 2023

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

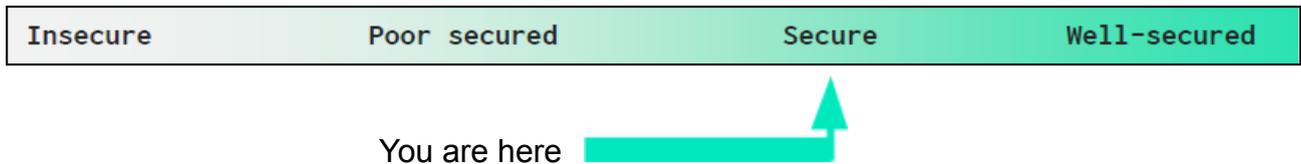
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
<p>File 1 Bridge.sol</p> <ul style="list-style-type: none"> • Bridge is used for inter-blockchain assets exchange • Owner can update extra coins rewards 	<p>YES, This is valid.</p> <p>Both Bridge and Pegged Tokens are Centralized solutions. Owner must handle them very securely.</p>
<p>File 2 PeggedToken.sol</p> <ul style="list-style-type: none"> • Name: ETH, BNB, Matic • Symbol: ETH, BNB, Matic • Decimal: 18 • Total Supply: 10 Million • No max minting limit. 	
<p>File 3 Proposal.sol</p> <ul style="list-style-type: none"> • Validators will vote to reactivate any inactive validator 	<p>YES, This is valid.</p>
<p>File 4 Punish.sol</p> <ul style="list-style-type: none"> • The validator can be punished for misbehavior • Validators can clean other validator's punish records while restake 	<p>YES, This is valid.</p>
<p>File 5 Validators.sol</p> <ul style="list-style-type: none"> • Validators' contracts can initialize and punish validators • It distributes block rewards to all active validators • Maximum Validators: 21 • Minimal Staking Coin: 32 Coins • Minimum Validator Staking: 1 Million • Distribution Of Gas Fee Earned By Validator: <ul style="list-style-type: none"> ○ Staker: 45% ○ Validator: 5% ○ Burn: 10% ○ Smart contract creator: 40% ○ Burn Stop Amount: 100 Million ○ Extra Rewards Per Block: 1 Coin 	<p>YES, This is valid.</p>

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, 0 high, 0 medium and 0 low and some very low level issues. All the issues have been acknowledged in the contract 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	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	Moderated
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Moderated
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: **PASSED**

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 MainnetZ 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 MainnetZ Chain Protocol.

The MainnetZ 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 MainnetZ 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://mainnetz.io/> 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

Bridge.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	onlySigner	modifier	Passed	No Issue
4	changeSigner	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	acceptOwnership	write	Passed	No Issue
7	receive	external	Passed	No Issue
8	coinIn	external	Function input parameters lack of check	Refer Audit Findings
9	coinOut	external	Function input parameters lack of check	Refer Audit Findings
10	tokenIn	external	Function input parameters lack of check, Hard coded Values	Refer Audit Findings
11	tokenOut	external	Function input parameters lack of check	Refer Audit Findings
12	setExtraCoinsRewards	external	Spelling mistake, Function input parameters lack of check	Refer Audit Findings

PeggedToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	transferOwnership	internal	Passed	No Issue
7	getOwner	external	Passed	No Issue
8	decimals	external	Passed	No Issue
9	symbol	external	Passed	No Issue
10	name	external	Passed	No Issue
11	totalSupply	external	Passed	No Issue
12	balanceOf	external	Passed	No Issue

13	transfer	external	Passed	No Issue
14	allowance	external	Passed	No Issue
15	approve	external	Passed	No Issue
16	transferFrom	external	Passed	No Issue
17	mint	write	Unlimited minting	Refer Audit Findings
18	burn	write	Passed	No Issue
19	transfer	internal	Passed	No Issue
20	mint	internal	Passed	No Issue
21	burn	internal	Passed	No Issue
22	approve	internal	Passed	No Issue

Proposal.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	onlyValidator	modifier	Passed	No Issue
10	initialize	external	Critical operation lacks event log, Infinite loops possibility	Refer Audit Findings
11	createProposal	external	Passed	No Issue
12	voteProposal	external	access only Validator	No Issue
13	setUnpassed	external	access only Validators Contract	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	Critical operation lacks event log	Refer Audit Findings
12	punish	external	access only Miner	No Issue
13	decreaseMissedBlocksCounter	external	Infinite loops possibility	Refer Audit Findings
14	cleanPunishRecord	external	access only Initialized	No Issue
15	getPunishValidatorsLen	read	Passed	No Issue
16	getPunishRecord	read	Passed	No Issue

Validators.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	onlyNotRewarded	modifier	Passed	No Issue
10	onlyNotUpdated	modifier	Passed	No Issue
11	receive	external	Passed	No Issue
12	setContractCreator	write	Critical operation lacks event log	Refer Audit Findings
13	initialize	external	Critical operation lacks event log, Infinite loops possibility	Refer Audit Findings
14	stake	write	access only Initialized	No Issue
15	createOrEditValidator	external	access only Initialized	No Issue
16	tryReactive	external	access only Proposal Contract	No Issue
17	unstake	external	access only Initialized	No Issue
18	withdrawStakingReward	write	Passed	No Issue
19	withdrawStaking	external	Passed	No Issue
20	withdrawProfits	external	Passed	No Issue
21	distributeBlockReward	external	Passed	No Issue
22	updateActiveValidatorSet	write	access only Miner	No Issue

23	removeValidator	external	access only Punish Contract	No Issue
24	removeValidatorIncoming	external	access only Punish Contract	No Issue
25	getValidatorDescription	read	Passed	No Issue
26	getValidatorInfo	read	Passed	No Issue
27	getStakingInfo	read	Passed	No Issue
28	getActiveValidators	read	Passed	No Issue
29	getTotalStakeOfActiveValidators	read	Passed	No Issue
30	getTotalStakeOfActiveValidators Except	read	Passed	No Issue
31	isActiveValidator	read	Passed	No Issue
32	isTopValidator	read	Passed	No Issue
33	getTopValidators	read	Passed	No Issue
34	validateDescription	write	Passed	No Issue
35	tryAddValidatorToHighestSet	internal	Passed	No Issue
36	tryRemoveValidatorIncoming	write	Passed	No Issue
37	addProfitsToActiveValidatorsByStakePercentExcept	write	Passed	No Issue
38	tryJailValidator	write	Passed	No Issue
39	tryRemoveValidatorInHighestSet	write	Passed	No Issue
40	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 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.

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

No low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Spelling mistake:

Validators.sol

```
// stake at first time to this valiador
if (staked[staker][validator].coins == 0) {
```

Spelling mistakes in comments.

“valiador” word should be “validator.”

Bridge.sol

```
function setExtraCoinsRewards(uint256 _extraCoinRewards) external onlyOwner returns( string memory){
    extraCoinRewards = _extraCoinRewards;
    return "Extra coins rewards updated";
}
```

Spelling mistakes in function name and parameter.

“setExtraCoinsRewards” should be “setExtraCoinsRewards”.

“_extraCoinRewards” should be “_extraCoinRewards”.

Resolution: Correct the spelling.

Status: This issue is Acknowledged by the MainnetZ team, as it does not impact any security or logical flow.

(2) Critical operation lacks event log:

Missing event log for:

Validators.sol

- initialize()
- setContractCreator()

Proposal.sol

- initialize()

Punish.sol

- cleanPunishRecord()

Resolution: Please write an event log for listed events.

Status: This issue is Acknowledged by the MainnetZ team, as these smart contracts are being called from the blockchain and no events are needed.

(3) Infinite loops possibility:

Validators.sol

```
// distributeBlockReward distributes block reward to all active validators
function distributeBlockReward(address[] memory _to, uint64[] memory _gass)
    external
    payable
    onlyMiner
    onlyNotRewarded
    onlyInitialized
{
    operationsDone[block.number][uint8(Operations.Distribute)] = true;
    address val = msg.sender;
    uint256 reward = msg.value;
    uint256 remaining = reward;
    //to validator
    uint _validatorPart = reward * validatorPartPercent / 100000;
    remaining = remaining - _validatorPart;

    //to burn
    uint _burnPart = reward * burnPartPercent / 100000;
    if(totalBurnt + _burnPart <= burnStopAmount )
    {
        remaining = remaining - _burnPart;
        totalBurnt += _burnPart;
        if(_burnPart > 0) payable(address(0)).transfer(_burnPart);
    }
    // to contract
    //uint _contractPart = reward * contractPartPercent / 100000;
    for (uint i=0; i<_to.length; i++)
    {
```

```
function initialize(address[] calldata vals) external onlyNotInitialized {  infinite gas
    punish = Punish(PunishContractAddr);

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

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.

Status: This issue is Acknowledged by the MainnetZ team, as validators are going to be no more than 21. So, it will never go above the limit.

(4) Function input parameters lack of check: [Bridge.sol](#)

Some functions require validation before execution.

Functions are:

- setExtraCoinsRewards
- coinOut
- coinIn
- tokenOut
- tokenIn

Resolution: We suggest using validation like for numerical variables that should be greater than 0 and for address type check variables that are not address(0).

Status: This issue is Acknowledged in the contract code.

(5) Hard coded Values: [Bridge.sol](#)

```
function tokenIn(address tokenAddress, uint256 tokenAmount, uint256 chainID, address outputCurrenc
orderID++;
//fund will go to the owner
if(tokenAddress == address(0xdAC17F958D2ee523a2206206994597C13D831ec7)){
    //There should be different interface for the USDT Ethereum contract
    usdtContract(tokenAddress).transferFrom(msg.sender, owner, tokenAmount);
}else{
    ERC20Essential(tokenAddress).transferFrom(msg.sender, owner, tokenAmount);
}
```

Some variables are set as hard coded addresses.

Resolution: Deployer has to confirm before deploying the contract to production.

Status: This issue is Acknowledged by the MainnetZ team, as this address is USDT Ethereum, which is a constant and the code logic is specifically for that.

(6) Unlimited minting: [PeggedToken.sol](#)

```
*/  
function _mint(address account, uint256 amount) internal { infinite gas  
    require(account != address(0), "BEP20: mint to the zero address");  
  
    _totalSupply = _totalSupply + amount;  
    _balances[account] = _balances[account] + amount;  
    emit Transfer(address(0), account, amount);  
}
```

Setting max minting for the tokens is good for tokenomics.

Resolution: Since this is an owner function, the owner must take care of minting with limitations. or even better, just add a max minting limit.

Status: This issue is Acknowledged by the MainnetZ team, as this is a centralized solution and the owner has the full ability to min or burn the tokens.

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:

- coinOut function in the Bridge contract: the signers can transfer coins out.
- tokenOut function in the Bridge contract: the signers can token out.
- setExtraCoinsRewards in the Bridge: the owner can update extra coins rewards.
- mint function in PeggedToken contact: the owner can create `amount` tokens and assign them to `msg.sender`, increasing the total supply.

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 not observed any major issue in the smart contracts. All the issues have been acknowledged in the contract code. **So smart contracts are 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”**.

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.

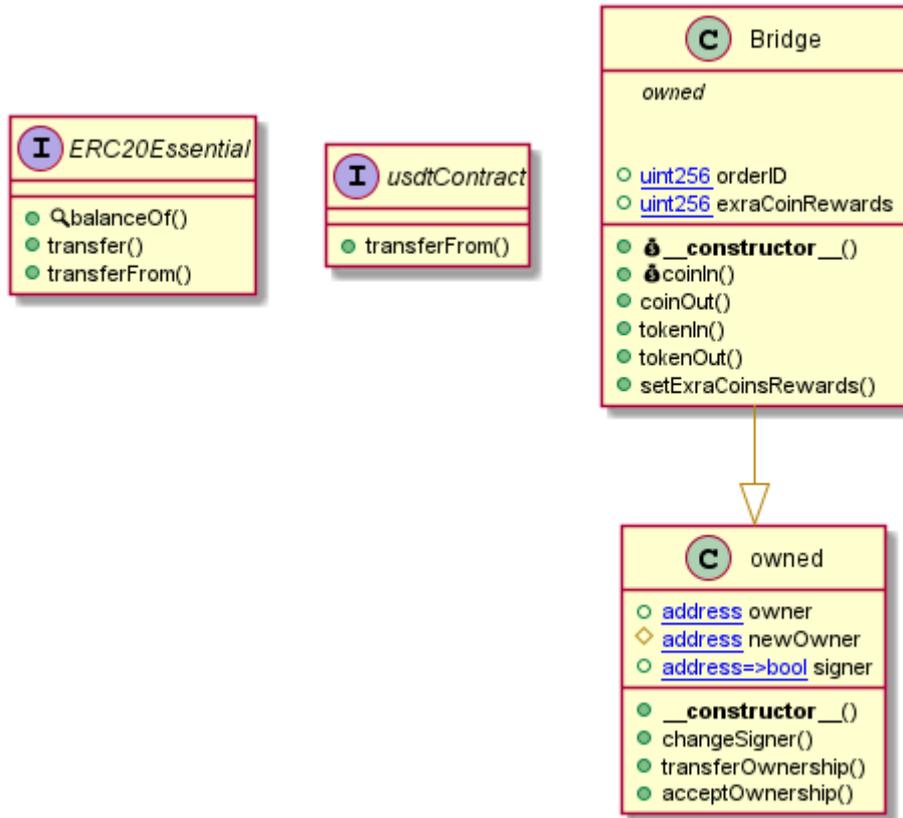
Technical Disclaimer

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.

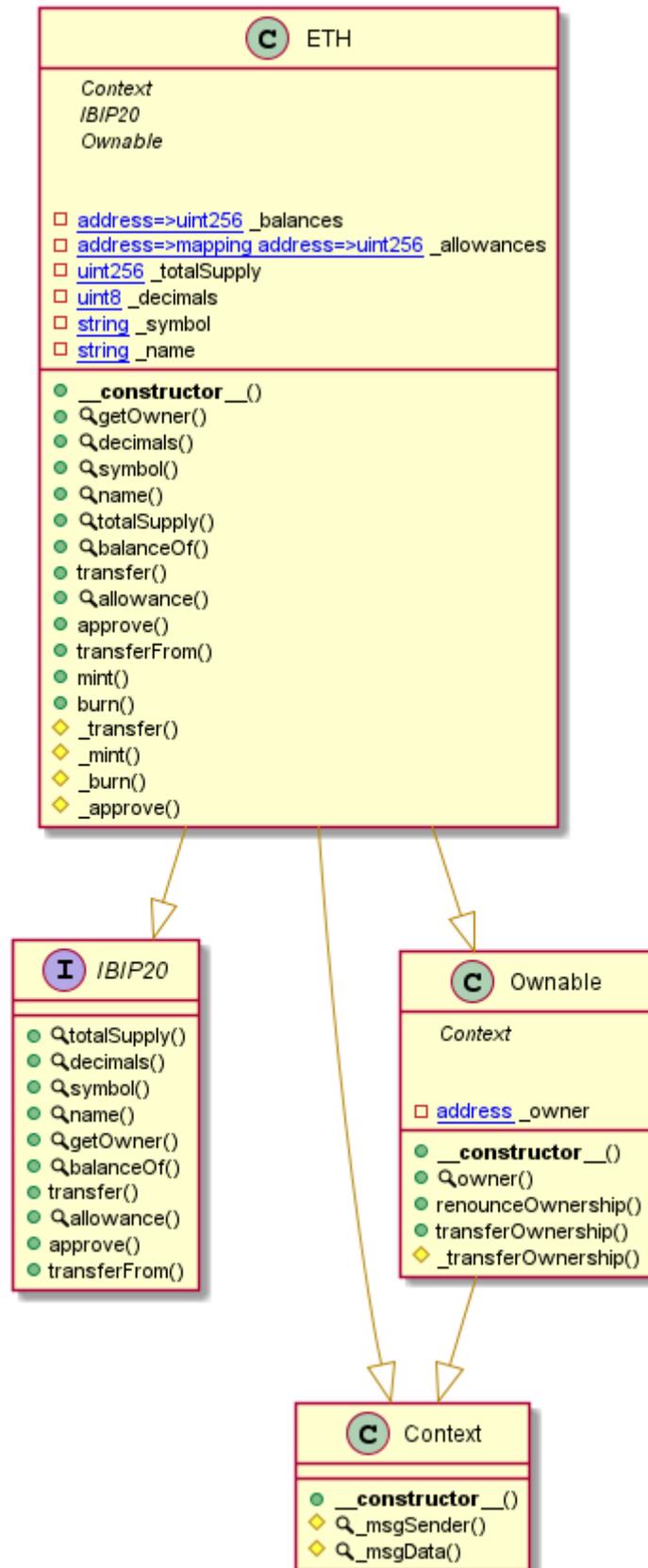
Appendix

Code Flow Diagram - MainnetZ Chain Protocol

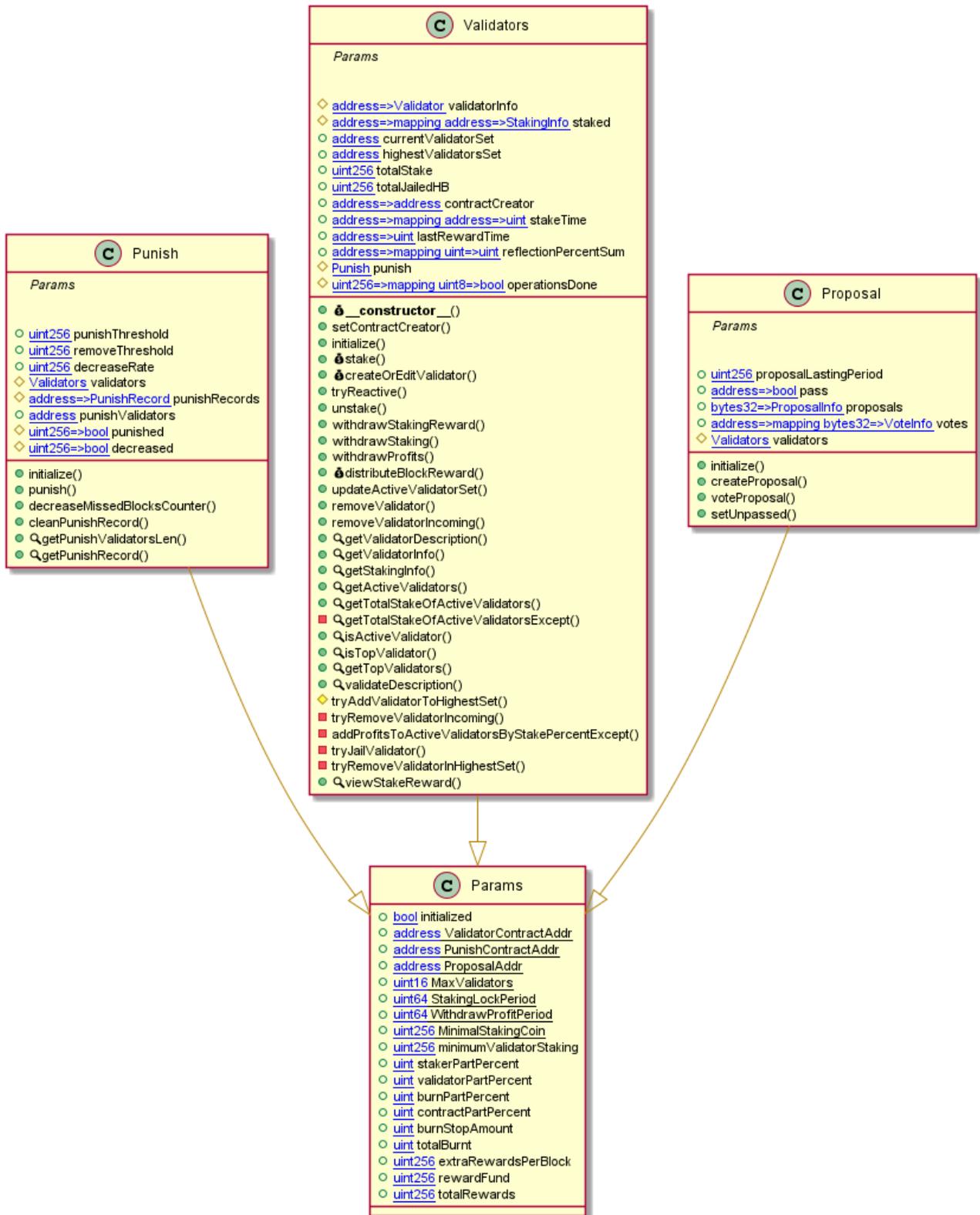
Bridge Diagram



PeggedToken Diagram



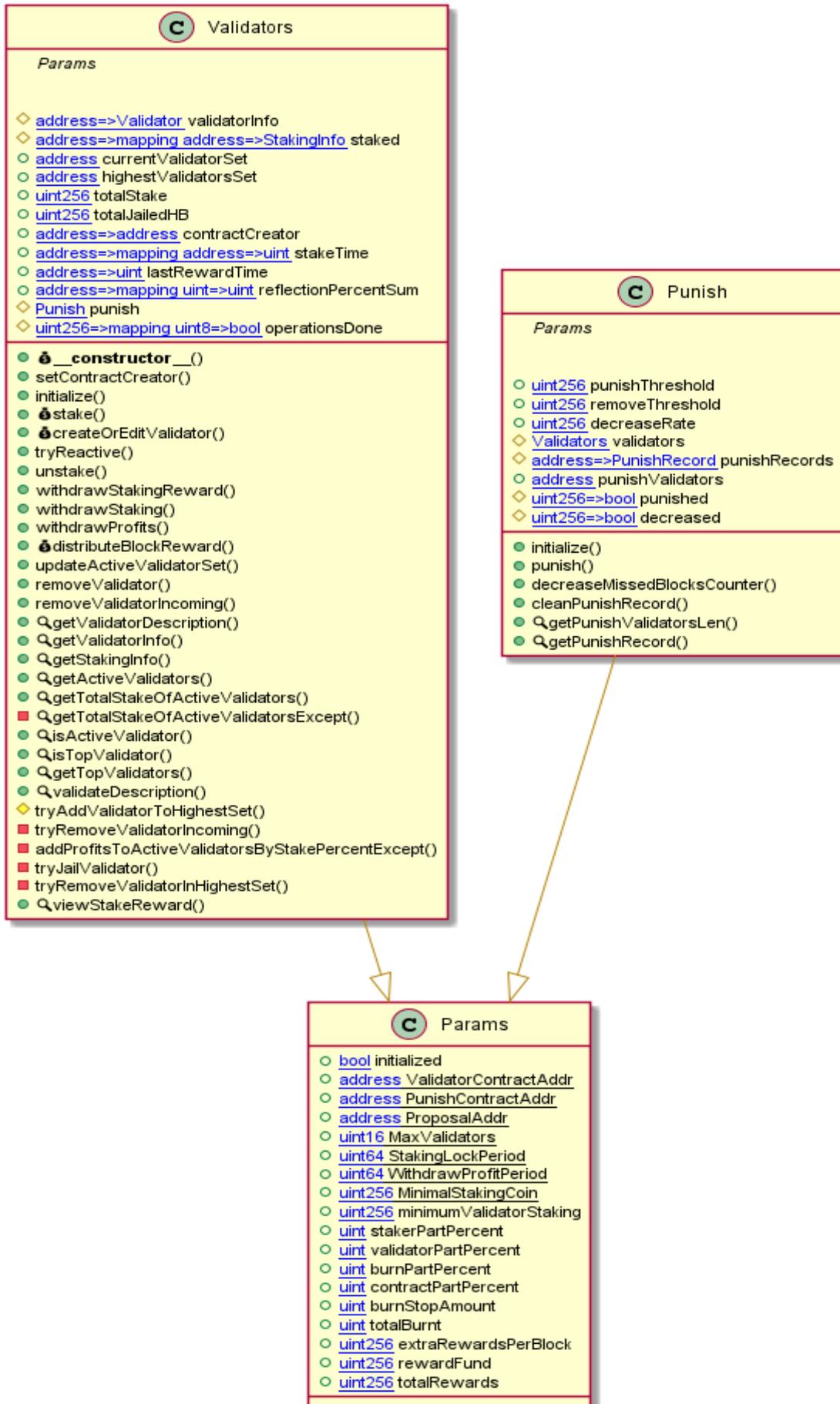
Proposal Diagram



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

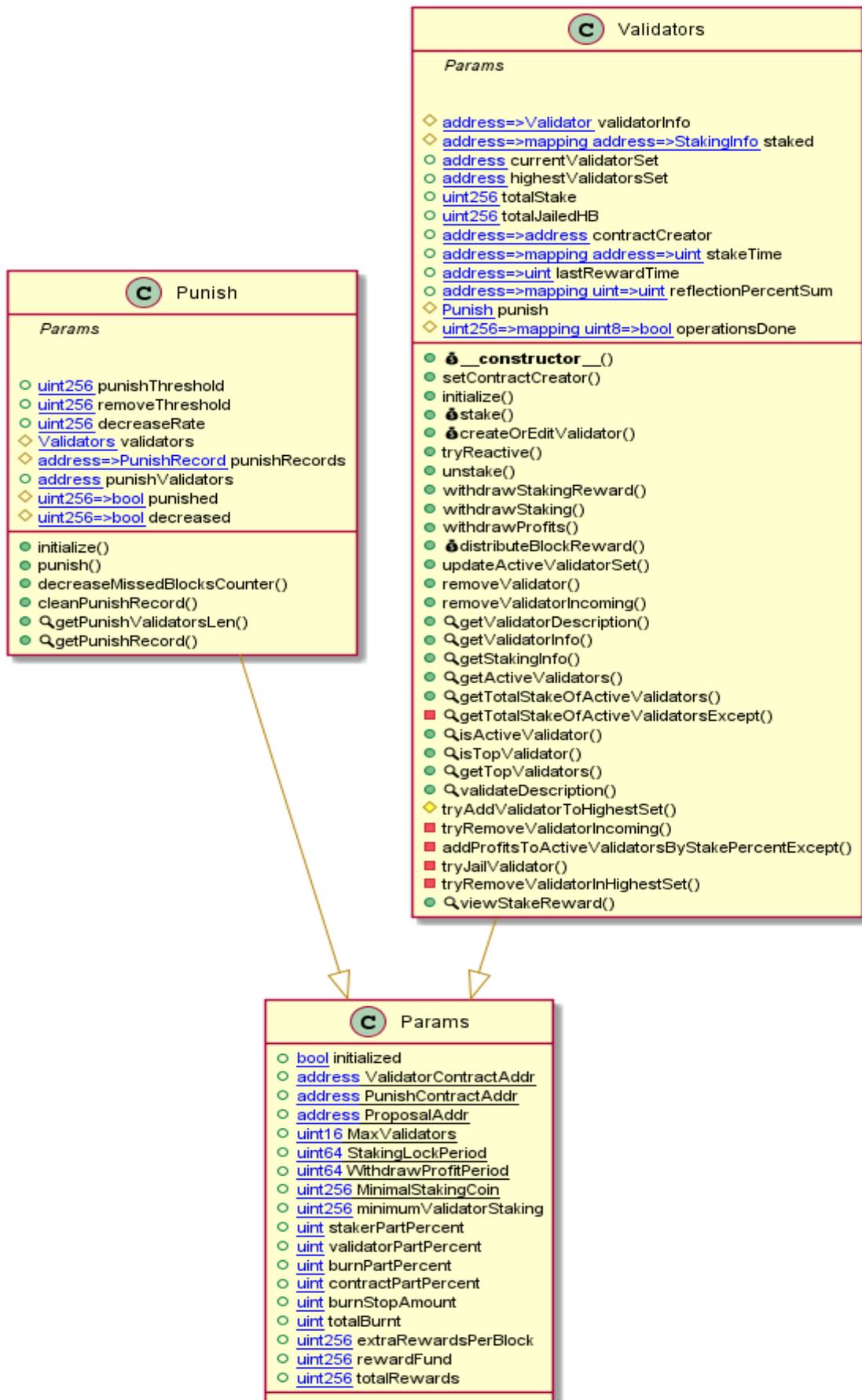
Punish Diagram



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Validators Diagram



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither Log >> Bridge.sol

```
INFO:Detectors:
Bridge.setExtraCoinsRewards(uint256) (Bridge.sol#137-140) should emit an event for:
- extraCoinRewards = _extraCoinRewards (Bridge.sol#138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
owned.transferOwnership(address)._newOwner (Bridge.sol#52) lacks a zero-check on:
- newOwner = _newOwner (Bridge.sol#53)
Bridge.coinOut(address,uint256,uint256).user (Bridge.sol#101) lacks a zero-check on:
- address(user).transfer(amount) (Bridge.sol#103)
Bridge.tokenOut(address,address,uint256,uint256,uint256).user (Bridge.sol#124) lacks a zero-check on:
- address(user).transfer(extraCoinRewards) (Bridge.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Bridge.tokenIn(address,uint256,uint256,address) (Bridge.sol#110-121):
External calls:
- usdtContract(tokenAddress).transferFrom(msg.sender,owner,tokenAmount) (Bridge.sol#115)
- ERC20Essential(tokenAddress).transferFrom(msg.sender,owner,tokenAmount) (Bridge.sol#117)
Event emitted after the call(s):
- TokenIn(orderID,tokenAddress,msg.sender,tokenAmount,chainID,outputCurrency) (Bridge.sol#119)
Reentrancy in Bridge.tokenOut(address,address,uint256,uint256,uint256) (Bridge.sol#124-134):
External calls:
- ERC20Essential(tokenAddress).transfer(user,tokenAmount) (Bridge.sol#126)
External calls sending eth:
- address(user).transfer(extraCoinRewards) (Bridge.sol#129)
Event emitted after the call(s):
- TokenOut(_orderID,tokenAddress,user,tokenAmount,chainID) (Bridge.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
INFO:Detectors:
Pragma version0.8.4 (Bridge.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:
Contract usdtContract (Bridge.sol#15-18) is not in CapWords
Contract owned (Bridge.sol#21-63) is not in CapWords
Parameter owned.changeSigner(address,bool)._signer (Bridge.sol#47) is not in mixedCase
Parameter owned.changeSigner(address,bool)._status (Bridge.sol#47) is not in mixedCase
Parameter owned.transferOwnership(address)._newOwner (Bridge.sol#52) is not in mixedCase
Parameter Bridge.coinOut(address,uint256,uint256).orderID (Bridge.sol#101) is not in mixedCase
Parameter Bridge.tokenOut(address,address,uint256,uint256,uint256).orderID (Bridge.sol#124) is not in mixedCase
Parameter Bridge.setExtraCoinsRewards(uint256)._extraCoinRewards (Bridge.sol#137) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in Bridge.coinIn(address) (Bridge.sol#94-99):
External calls:
- address(owner).transfer(msg.value) (Bridge.sol#96)
Event emitted after the call(s):
- CoinIn(orderID,msg.sender,msg.value,outputCurrency) (Bridge.sol#97)
Reentrancy in Bridge.coinOut(address,uint256,uint256) (Bridge.sol#101-107):
External calls:
- address(user).transfer(amount) (Bridge.sol#103)
Event emitted after the call(s):
- CoinOut(_orderID,user,amount) (Bridge.sol#104)
Reentrancy in Bridge.tokenOut(address,address,uint256,uint256,uint256) (Bridge.sol#124-134):
External calls:
- address(user).transfer(extraCoinRewards) (Bridge.sol#129)
Event emitted after the call(s):
- TokenOut(_orderID,tokenAddress,user,tokenAmount,chainID) (Bridge.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
```

```
INFO:Detectors:
changeSigner(address,bool) should be declared external:
- owned.changeSigner(address,bool) (Bridge.sol#47-50)
transferOwnership(address) should be declared external:
- owned.transferOwnership(address) (Bridge.sol#52-54)
acceptOwnership() should be declared external:
- owned.acceptOwnership() (Bridge.sol#57-62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Bridge.sol analyzed (4 contracts with 75 detectors), 27 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither Log >> PeggedToken.sol

```
INFO:Detectors:
ETH.allowance(address,address).owner (PeggedToken.sol#285) shadows:
- Ownable.owner() (PeggedToken.sol#164-166) (function)
ETH._approve(address,address,uint256).owner (PeggedToken.sol#413) shadows:
- Ownable.owner() (PeggedToken.sol#164-166) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Context._msgData() (PeggedToken.sol#114-117) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.4 (PeggedToken.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:
Redundant expression "this (PeggedToken.sol#115)" inContext (PeggedToken.sol#105-118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Solidity Static Analysis

Bridge.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Bridge.coinIn(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 99:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Bridge.tokenOut(address,address,uint256,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 129:4:

Gas & Economy

Gas costs:

Gas requirement of function Bridge.coinOut 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: 106:4:

Gas costs:

Gas requirement of function Bridge.setExtraCoinsRewards 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: 142:4:

PeggedToken.sol

Gas & Economy

Gas costs:

Gas requirement of function ETH.symbol 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: 244:2:

Gas costs:

Gas requirement of function ETH.mint 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: 328:2:

Gas costs:

Gas requirement of function ETH.burn 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: 336:2:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

[more](#)

Pos: 13:2:

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

[more](#)

Pos: 237:2:

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: 366:37:

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: 1252:33:

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: 146:4:

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: 866:4:

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: 235:37:

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: 1018:37:

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: 239:4:

Gas costs:

Gas requirement of function Validators.distributeBlockReward 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: 529:4:

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: 366:37:

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: 146:4:

Gas costs:

Gas requirement of function Validators.distributeBlockReward 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: 660:4:

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: 1076:19:

Solhint Linter

Bridge.sol

```
Bridge.sol:2:1: Error: Compiler version 0.8.17 does not satisfy the r semver requirement
Bridge.sol:15:1: Error: Contract name must be in CamelCase
Bridge.sol:26:1: Error: Contract name must be in CamelCase
Bridge.sol:35:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Bridge.sol:47:37: Error: Use double quotes for string literals
Bridge.sol:95:33: Error: Code contains empty blocks
```

PeggedToken.sol

```
PeggedToken.sol:2:1: Error: Compiler version 0.8.17 does not satisfy the r semver requirement
PeggedToken.sol:108:3: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
PeggedToken.sol:108:19: Error: Code contains empty blocks
PeggedToken.sol:155:3: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
PeggedToken.sol:217:3: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
```

Proposal.sol

```
Proposal.sol:2:1: Error: Compiler version 0.8.17 does not satisfy the r semver requirement
Error: Constant name must be in capitalized SNAKE_CASE
Proposal.sol:13:25: Error: Constant name must be in capitalized SNAKE_CASE
Proposal.sol:89:5: Error: Explicitly mark visibility of state
Proposal.sol:143:38: Error: Avoid to make time-based decisions in your business logic
Proposal.sol:366:38: Error: Avoid to use tx.origin
Proposal.sol:375:39: Error: Avoid making time-based decisions in your
Proposal.sol:709:13: Error: Possible reentrancy vulnerabilities. Avoid state changes after transfer.
Proposal.sol:727:60: Error: Avoid to make time-based decisions in your business logic
Proposal.sol:1120:5: Error: Explicitly mark visibility of state
Proposal.sol:1171:56: Error: Avoid to make time-based decisions in your business logic
```

Punish.sol

```
Punish.sol:2:1: Error: Compiler version 0.8.17 does not satisfy the r
semver requirement
Punish.sol:9:25: Error: Constant name must be in capitalized
SNAKE_CASE
Punish.sol:11:25: Error: Constant name must be in capitalized
SNAKE_CASE
Punish.sol:21:29: Error: Constant name must be in capitalized
SNAKE_CASE
Punish.sol:208:5: Error: Event name must be in CamelCase
Punish.sol:235:38: Error: Avoid to use tx.origin
Punish.sol:244:39: Error: Avoid to make time-based decisions in your
business logic
Punish.sol:571:9: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
Punish.sol:571:31: Error: Avoid to make time-based decisions in your
business logic
Punish.sol:932:54: Error: Avoid to make time-based decisions in your
business logic
Punish.sol:964:5: Error: Explicitly mark visibility of state
Punish.sol:966:5: Error: Explicitly mark visibility of state
Punish.sol:969:5: Error: Explicitly mark visibility of state
Punish.sol:970:5: Error: Explicitly mark visibility of state
```

Validators.sol

```
Validators.sol:2:1: Error: Compiler version 0.8.17 does not satisfy
the r semver requirement
Validators.sol:20:28: Error: Constant name must be in capitalized
SNAKE_CASE
Validators.sol:21:29: Error: Constant name must be in capitalized
SNAKE_CASE
Validators.sol:95:5: Error: Explicitly mark visibility of state
Validators.sol:143:38: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:366:38: Error: Avoid to use tx.origin
Validators.sol:375:39: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:702:9: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
Validators.sol:702:31: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:957:50: Error: Avoid to make time-based decisions in
your business logic
Validators.sol:1063:54: 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.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io