Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	The Staking Carnival
Website:	https://carn.app
Platform:	Ethereum
Language:	Solidity
Date:	February 25th, 2023

Table of contents

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

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the PulseDogecoin Staking Carnival team to perform the Security audit of the Staking Carnival smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on February 25th, 2023.

The purpose of this audit was to address the following:

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

Project Background

- The Staking Carnival aims to provide a new staking mechanism that imbues $PLSD with utility, value and YIELD.
- The Staking Carnival is a decentralized finance (DeFi) platform that allows users to stake their tokens for 90-day periods. The staking period is standardized, with all users participating for the same 90 days. The only way to earn rewards is by staking for the entire 90-day period, which means that you need to stake before the 90-day period starts and end the stake after the 90-day period ends.
- The Staking Carnival contract inherits the IERC20, SafeERC20, ReentrancyGuard, ERC721, ERC721URIStorage, ERC721Burnable, Counters, Ownable, ERC20, ERC20Burnable standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	Code Review and Security Analysis Report for The Staking Carnival Smart Contracts
Platform	**Ethereum / Solidity**
File 1	CarnTokenBooth.sol
File 1 MD5 Hash	C282C8FD6967AB470ACBB518E238B684
File 2	CommunityCarnivalASICMiner.sol
File 2 MD5 Hash	D9358F0890DD0BC2F9BBCB9E284C66CF
File 3	PLSDStaker.sol
File 3 MD5 Hash	0C03EEE9DE96CFFE8D087A766CA89418
File 4	PulseBitcoinLockRewards.sol
File 4 MD5 Hash	97A1A9105586353A853FC3775A86A056
File 5	WaatcaNFT.sol
File 5 MD5 Hash	802747EB0E9237D59A66DE22A5EEAE91
Audit Date	February 25th,2023
Revised Audit Date	February 28th,2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 CarnTokenBooth.sol • Name: Carnival • Symbol: CARN • Decimals: 12 • Openzeppelin libraries are used.	**YES, This is valid.**
File 2 CommunityCarnivalASICMiner.sol • Mining Period: 30 Days • Reload Period: 5 Days • Trapped Pool Target: 100k CARN • Carn Cost: 10 CARN • Minimum Asic Deposit: 25 ASIC • Openzeppelin libraries are used.	**YES, This is valid.**
File 3 PLSDStaker.sol • The Staking allows players to stake $PLSD for 90 day periods, and get rewards in the form of $PLSD, $PLSB and $ASIC and much more.	**YES, This is valid.**
File 4 PulseBitcoinLockRewards.sol • The owner can set a carn address	**YES, This is valid.**
File 5 WaatcaNFT.sol • The owner can set a carn address. **Other Specifications:** • WAATCA NFTs allow users to passively benefit from all rewards generated at the CARNival. Ownership of a WAATCA NFT is like holding a share in the entire CARNival.	**YES, This is valid.**

Audit Summary

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

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 low and some very low level issues.
All these issues are resolved in the revised 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	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“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 The Staking Carnival Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the The Staking Carnival Protocol.

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

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

Documentation

We were given the Staking Carnival smart contract code in the form of a private github repository. The hashes of that code are mentioned above in the table.

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

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

Use of Dependencies

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

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

AS-IS overview

CarnTokenBooth.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	nonReentrant	modifier	Passed	No Issue
3	_nonReentrantBefore	write	Passed	No Issue
4	_nonReentrantAfter	write	Passed	No Issue
5	_reentrancyGuardEntered	internal	Passed	No Issue
6	burn	write	Passed	No Issue
7	burnFrom	write	Passed	No Issue
8	buyCARN	write	Passed	No Issue
9	decimals	read	Passed	No Issue
10	getOwner	external	Passed	No Issue
11	name	read	Passed	No Issue
12	decimals	write	Passed	No Issue
13	symbol	read	Passed	No Issue
14	totalSupply	read	Passed	No Issue
15	balanceOf	read	Passed	No Issue
16	transfer	write	Passed	No Issue
17	allowance	write	Passed	No Issue
18	approve	write	Passed	No Issue
19	transferFrom	write	Passed	No Issue
20	increaseAllowance	write	Passed	No Issue
21	decreaseAllowance	write	Passed	No Issue
22	_spendAllowance	internal	Passed	No Issue
23	_transfer	internal	Passed	No Issue
24	_mint	internal	Passed	No Issue
25	_burn	internal	Passed	No Issue
26	_approve	internal	Passed	No Issue
27	_afterTokenTransfer	internal	Passed	No Issue
28	_beforeTokenTransfer	internal	Passed	No Issue

CommunityCarnivalASICMiner.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	deposit	write	Passed	No Issue
3	startMiningSession	write	Passed	No Issue
4	getMinerStore	internal	Passed	No Issue
5	startReloadPeriod	write	Passed	No Issue
6	claimReward	external	Passed	No Issue
7	depositCARNToTrappedPool	external	Passed	No Issue
8	releaseASIC	internal	Same validation performed	Refer Audit Findings

9	releaseCARN	internal	Same validation performed	Refer Audit Findings
10	nonReentrant	modifier	Passed	No Issue
11	_nonReentrantBefore	write	Passed	No Issue
12	_nonReentrantAfter	write	Passed	No Issue
13	_reentrancyGuardEntered	internal	Passed	No Issue

PLSDStaker.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	stake	write	Passed	No Issue
3	startLockedPeriod	write	Passed	No Issue
4	startLockingPeriod	write	Passed	No Issue
5	emergencyEnd	write	Passed	No Issue
6	depositPLSD	write	Passed	No Issue
7	depositPLSB	write	Passed	No Issue
8	depositASIC	write	Passed	No Issue
9	depositHEX	write	Passed	No Issue
10	claimRewards	write	Passed	No Issue
11	nonReentrant	modifier	Passed	No Issue
12	_nonReentrantBefore	write	Passed	No Issue
13	_nonReentrantAfter	write	Passed	No Issue
14	_reentrancyGuardEntered	internal	Passed	No Issue

PulseBitcoinLockRewards.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	setCarnAddress	write	access only Owner	No Issue
3	withdrawRewards	write	Spelling mistake	Fixed
4	registerNftForRewards	write	Passed	No Issue
5	currentDay	external	Passed	No Issue
6	_currentDay	internal	Passed	No Issue
7	owner	read	Passed	No Issue
8	onlyOwner	modifier	Passed	No Issue
9	renounceOwnership	write	access only Owner	No Issue
10	transferOwnership	write	access only Owner	No Issue

WaatcaNFT.sol

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	setCarnAddress	write	access only Owner	No Issue
7	mintWaatcaNft	write	Passed	No Issue
8	mint	internal	Passed	No Issue
9	tokenURI	read	Passed	No Issue
10	_burn	internal	Passed	No Issue
11	supportsInterface	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	ownerOf	read	Passed	No Issue
14	name	read	Passed	No Issue
15	symbol	read	Passed	No Issue
16	tokenURI	read	Passed	No Issue
17	_baseURI	internal	Passed	No Issue
18	approve	write	Passed	No Issue
19	getApproved	read	Passed	No Issue
20	setApprovalForAll	write	Passed	No Issue
21	isApprovedForAll	read	Passed	No Issue
22	transferFrom	write	Passed	No Issue
23	safeTransferFrom	write	Passed	No Issue
24	safeTransferFrom	write	Passed	No Issue
25	_safeTransfer	internal	Passed	No Issue
26	_ownerOf	internal	Passed	No Issue
27	_exists	internal	Passed	No Issue
28	_isApprovedOrOwner	internal	Passed	No Issue
29	_safeMint	internal	Passed	No Issue
30	_mint	internal	Passed	No Issue
31	_burn	internal	Passed	No Issue
32	_transfer	internal	Passed	No Issue
33	_approve	internal	Passed	No Issue
34	_setApprovalForAll	internal	Passed	No Issue
35	_requireMinted	internal	Passed	No Issue
36	_checkOnERC721Received	write	Passed	No Issue
37	_beforeTokenTransfer	internal	Passed	No Issue
38	_afterTokenTransfer	internal	Passed	No Issue
39	_beforeConsecutiveTokenTransfer	internal	Passed	No Issue
40	_afterConsecutiveTokenTransfer	internal	Passed	No Issue

Severity Definitions

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

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Same validation performed: **CommunityCarnivalASICMiner.sol**

```
function depositCARNToTrappedPool(uint256 _carnAmount) external nonReentrant {
    trappedAsicReleasePool += _carnAmount;
    IERC20(CARN).safeTransferFrom(msg.sender, address(this), _carnAmount);
    emit CARNDepositToTrappedPool(msg.sender, _carnAmount, block.timestamp);
    if (
        trappedAsicReleasePool >= TRAPPED_POOL_TARGET &&
        state == State.RELOAD
    ) {
        releaseASIC();
        releaseCARN();
        trappedAsicReleasePool = 0;
    }
}

function releaseASIC() internal {
    require(trappedAsicReleasePool >= TRAPPED_POOL_TARGET, "Target not reached yet");
    uint256 _asicBalance = IERC20(ASIC).balanceOf(address(this)) - totalAsicDepositForTheCurrentSession;
    uint256 _asicToPlsdStaker = (_asicBalance * 60) / 100;
    uint256 _asicToWaatca = _asicBalance - _asicToPlsdStaker;

    IERC20(ASIC).approve(plsdStakingContract, _asicToPlsdStaker);
    PLSDStaker(plsdStakingContract).depositASIC(_asicToPlsdStaker);
    IERC20(ASIC).safeTransfer(waatcaPool, _asicToWaatca);
    emit ASICReleased(_asicBalance, block.timestamp);
}

function releaseCARN() internal {
    require(trappedAsicReleasePool >= TRAPPED_POOL_TARGET, "Target not reached yet");
    uint256 _carnBalance = IERC20(CARN).balanceOf(address(this));
    IERC20(CARN).safeTransfer(buyAndBurnContract, _carnBalance);
    emit CARNReleased(_carnBalance, block.timestamp);
}
```

“releaseASIC” and “releaseCARN” are internal functions which are used to execute from the “depositCARNToTrappedPool” function only when the trappedAsicReleasePool is greater than or equal to the Trapped pool target. This same validation has been performed inside both internal functions.

Resolution: We suggest removing the same validation from internal functions to save some gas fee.

Status: This issue is resolved in revised contract code.

Very Low / Informational / Best practices:

(1) Spelling mistake: **PulseBitcoinLockNFTRewards.sol**

```
function withdrawRewards(uint256 tokenId) public {
    require(
        msg.sender == pulseBitcoinLockNftContract.ownerOf(tokenId),
        "You are not the owner of this NFT"
    );
    require(
        tokenIdsToRegistered[tokenId],
        "You must register your NFT for rewards first"
    );
    require(
        tokenIdsToLastWithdrawalDay[tokenId] <
            tokenIdsToEndRewardsDay[tokenId],
        "You have already recieved all possible rewards for this NFT"
    );
```

Incorrect spelling of received in “withdrawRewards” function.

Resolution: We suggest correcting the spelling.

Status: This issue is resolved in revised contract code.

(2) Unused function: **CommunityCarnivalASICMiner.sol**

This contract includes the interface PulseBitcoin in which minerList() has been defined twice with different parameters, the one which is defined with a single parameter is wrong and unused.

Resolution: We suggest removing unused functions.

Status: This issue is resolved in 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:

PulseBitcoinLockNFTRewards.sol

- setCarnAddress: The owner can set a carn address.

WaatcaNFT.sol

- setCarnAddress: The owner can set a carn address.

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 low issue and some Informational severity issues in the smart contracts. All issues have been fixed / acknowledged in the 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 **"Secured"**.

Email: audit@EtherAuthority.io

Our Methodology

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

Manual Code Review:

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

Vulnerability Analysis:

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

Documenting Results:

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

Suggested Solutions:

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

Disclaimers

EtherAuthority.io Disclaimer

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

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

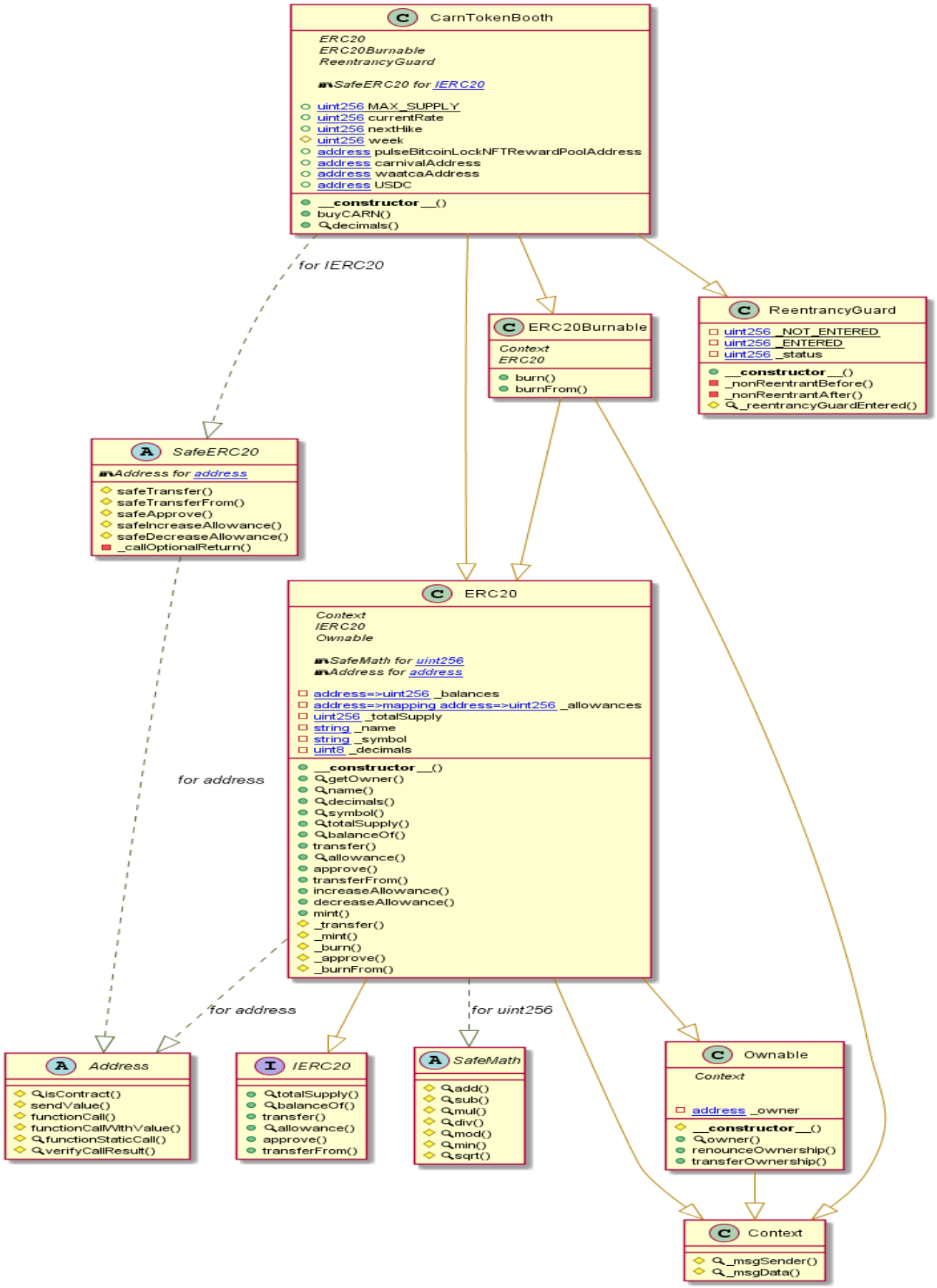
Technical Disclaimer

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

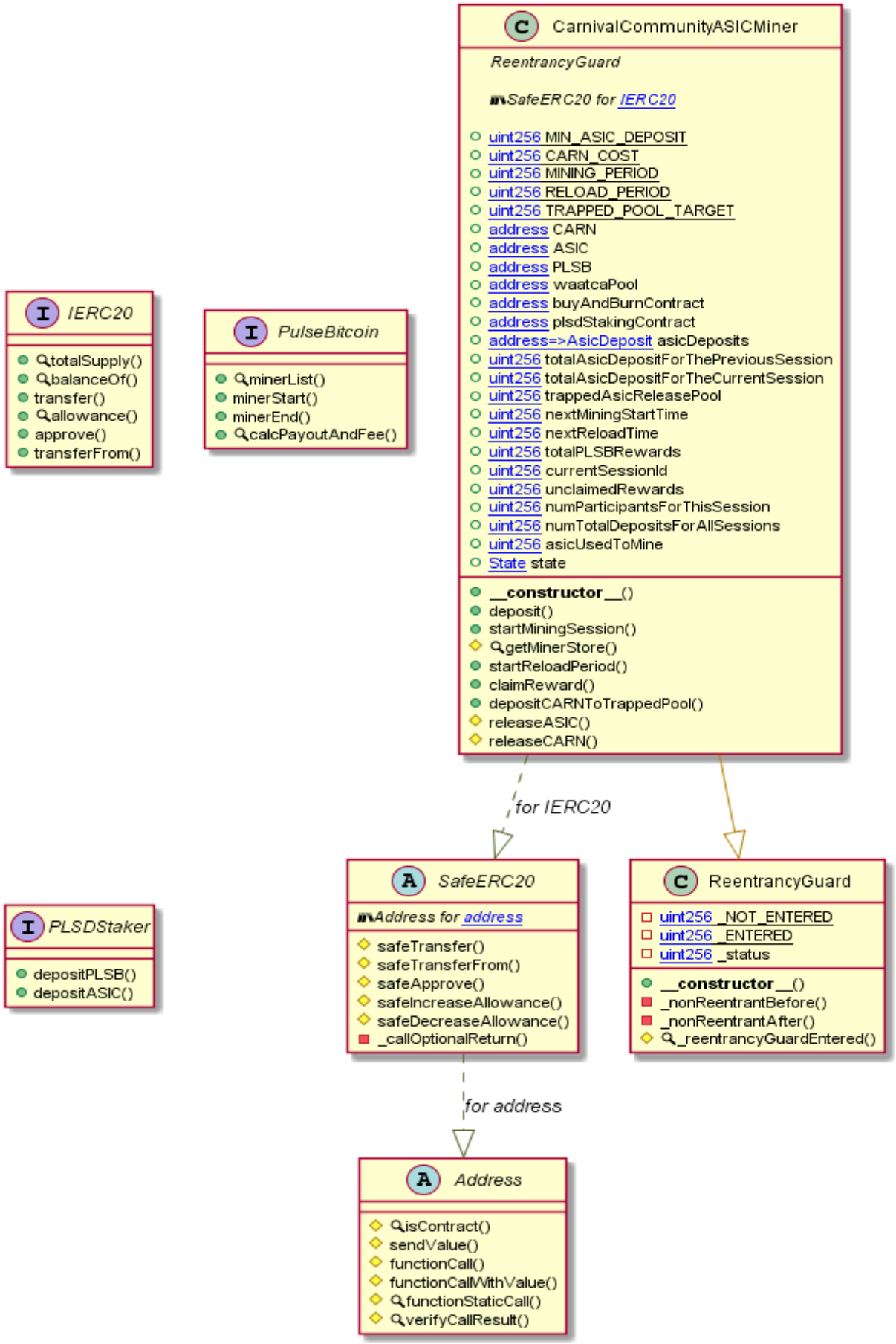
Appendix

Code Flow Diagram - The PulseDogecoin Staking Carnival

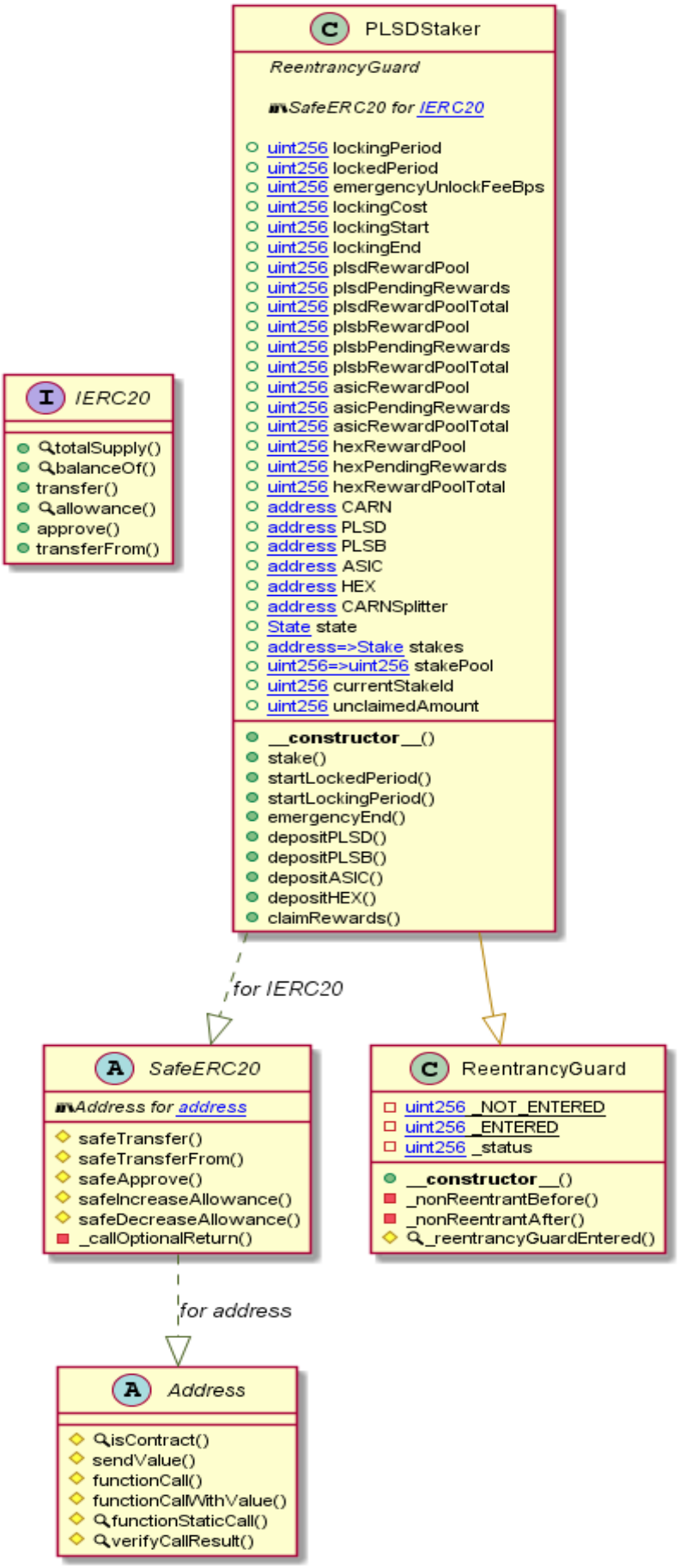
CarnTokenBooth Diagram



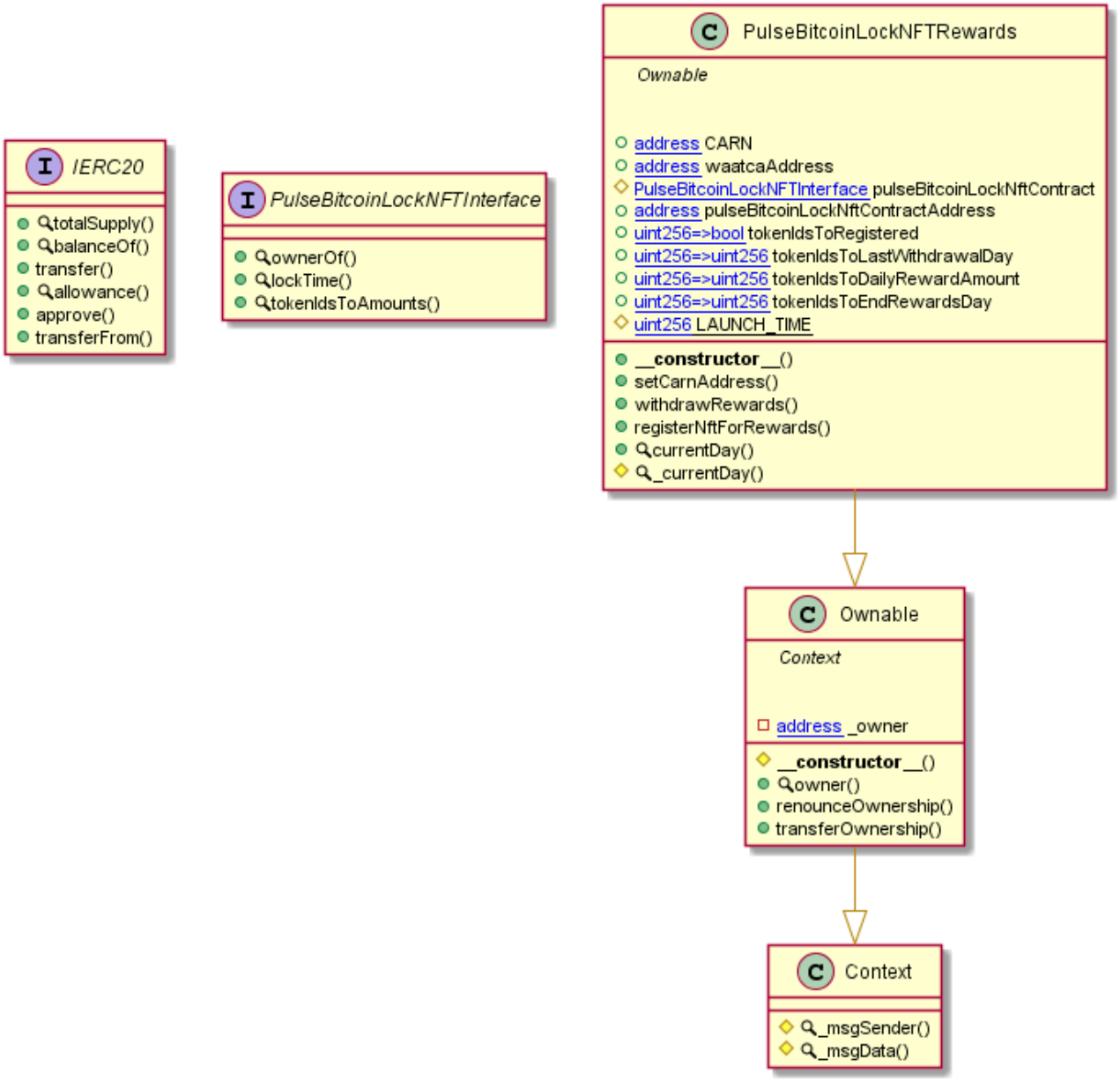
CommunityCarnivalASICMiner Diagram



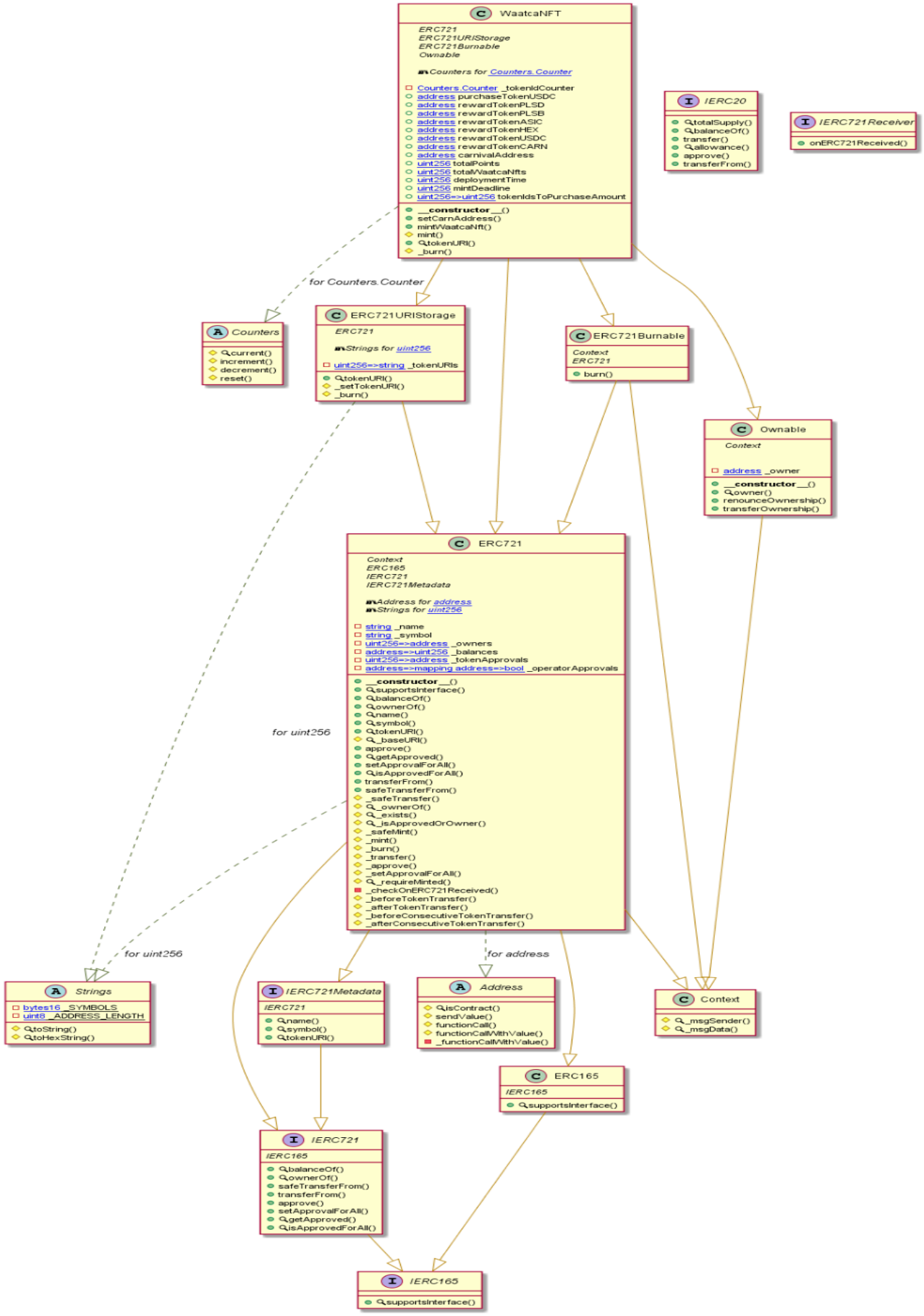
PLSDStaker Diagram



PulseBitcoinLockRewards Diagram



WaatcaNFT Diagram



Email: audit@EtherAuthority.io

Slither Results Log

Slither log >> CarnTokenBooth.sol

```
CarnTokenBooth.constructor(address,address,address,address)._pulseBitcoinLockNFTRewardPoolAddress (CarnTokenBooth.sol#996) lac
ks a zero-check on :
                - pulseBitcoinLockNFTRewardPoolAddress = _pulseBitcoinLockNFTRewardPoolAddress (CarnTokenBooth.sol#1001)
CarnTokenBooth.constructor(address,address,address,address)._carnivalAddress (CarnTokenBooth.sol#997) lacks a zero-check on :
                - carnivalAddress = _carnivalAddress (CarnTokenBooth.sol#1002)
CarnTokenBooth.constructor(address,address,address,address)._waatcaAddress (CarnTokenBooth.sol#998) lacks a zero-check on :
                - waatcaAddress = _waatcaAddress (CarnTokenBooth.sol#1003)
CarnTokenBooth.constructor(address,address,address,address)._USDC (CarnTokenBooth.sol#999) lacks a zero-check on :
                - USDC = _USDC (CarnTokenBooth.sol#1004)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in CarnTokenBooth.buyCARN(uint256) (CarnTokenBooth.sol#1011-1032):
        External calls:
        - IERC20(USDC).safeTransferFrom(msg.sender,carnivalAddress,_usdcValue / 2) (CarnTokenBooth.sol#1018-1022)
        - IERC20(USDC).safeTransferFrom(msg.sender,waatcaAddress,_usdcValue / 2) (CarnTokenBooth.sol#1023-1027)
        State variables written after the call(s):
        - _mint(msg.sender,amount) (CarnTokenBooth.sol#1031)
                - _balances[account] = _balances[account].add(amount) (CarnTokenBooth.sol#835)
        - _mint(msg.sender,amount) (CarnTokenBooth.sol#1031)
                - _totalSupply = _totalSupply.add(amount) (CarnTokenBooth.sol#834)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
Reentrancy in CarnTokenBooth.buyCARN(uint256) (CarnTokenBooth.sol#1011-1032):
        External calls:
        - IERC20(USDC).safeTransferFrom(msg.sender,carnivalAddress,_usdcValue / 2) (CarnTokenBooth.sol#1018-1022)
        - IERC20(USDC).safeTransferFrom(msg.sender,waatcaAddress,_usdcValue / 2) (CarnTokenBooth.sol#1023-1027)
        State variables written after the call(s):
        - _mint(msg.sender,amount) (CarnTokenBooth.sol#1031)
                - _balances[account] = _balances[account].add(amount) (CarnTokenBooth.sol#835)
        - _mint(msg.sender,amount) (CarnTokenBooth.sol#1031)
                - _totalSupply = _totalSupply.add(amount) (CarnTokenBooth.sol#834)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

CarnTokenBooth.buyCARN(uint256) (CarnTokenBooth.sol#1011-1032) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= nextHike (CarnTokenBooth.sol#1012)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.verifyCallResult(bool,bytes,string) (CarnTokenBooth.sol#271-291) uses assembly
        - INLINE ASM (CarnTokenBooth.sol#283-286)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

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

Low level call in Address.sendValue(address,uint256) (CarnTokenBooth.sol#130-135):
        - (success) = recipient.call{value: amount}() (CarnTokenBooth.sol#133)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (CarnTokenBooth.sol#198-209):
        - (success,returndata) = target.call{value: value}(data) (CarnTokenBooth.sol#207)
Low level call in Address.functionStaticCall(address,bytes,string) (CarnTokenBooth.sol#227-236):
        - (success,returndata) = target.staticcall(data) (CarnTokenBooth.sol#234)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter CarnTokenBooth.buyCARN(uint256)._usdcValue (CarnTokenBooth.sol#1011) is not in mixedCase
Variable CarnTokenBooth.USDC (CarnTokenBooth.sol#993) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (CarnTokenBooth.sol#556)" inContext (CarnTokenBooth.sol#550-559)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
CarnTokenBooth.sol analyzed (10 contracts with 84 detectors), 37 result(s) found
```

Slither log >> CommunityCarnivalASICMiner.sol

```
CarnivalCommunityASICMiner.deposit(uint256) (CommunityCarnivalASICMiner.sol#560-605) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > nextReloadTime && state != State.RELOAD (CommunityCarnivalASICMiner.sol#563)
        - block.timestamp > nextMiningStartTime && state == State.RELOAD (CommunityCarnivalASICMiner.sol#602)
CarnivalCommunityASICMiner.startMiningSession() (CommunityCarnivalASICMiner.sol#607-636) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > nextMiningStartTime,Reload period not ended yet) (CommunityCarnivalASICMiner.
sol#609-612)
CarnivalCommunityASICMiner.startReloadPeriod() (CommunityCarnivalASICMiner.sol#642-671) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > nextReloadTime,Mining session not ended yet) (CommunityCarnivalASICMiner.sol#
645-648)
CarnivalCommunityASICMiner.claimReward() (CommunityCarnivalASICMiner.sol#673-706) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > nextReloadTime && state != State.RELOAD (CommunityCarnivalASICMiner.sol#674)
        - block.timestamp > nextMiningStartTime && state == State.RELOAD (CommunityCarnivalASICMiner.sol#702)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Reentrancy in CarnivalCommunityASICMiner.startMiningSession() (CommunityCarnivalASICMiner.sol#607-636):
        External calls:
        - PulseBitcoin(PLSB).minerStart(_asicBalance) (CommunityCarnivalASICMiner.sol#630)
        Event emitted after the call(s):
        - MiningSessionStart(msg.sender,currentSessionId - 1,block.timestamp) (CommunityCarnivalASICMiner.sol#631-635)
Reentrancy in CarnivalCommunityASICMiner.startReloadPeriod() (CommunityCarnivalASICMiner.sol#642-671):
        External calls:
        - PulseBitcoin(PLSB).minerEnd(0,_minerId,address(this)) (CommunityCarnivalASICMiner.sol#658)
        - IERC20(PLSB).approve(plsdStakingContract,_plsbToTransfer) (CommunityCarnivalASICMiner.sol#660)
        - PLSDStaker(plsdStakingContract).depositPLSB(_plsbToTransfer) (CommunityCarnivalASICMiner.sol#661)
        - IERC20(PLSB).safeTransfer(waatcaPool,_plsbToTransfer) (CommunityCarnivalASICMiner.sol#662)
        - IERC20(ASIC).approve(plsdStakingContract,_asicToTransfer) (CommunityCarnivalASICMiner.sol#664)
        - PLSDStaker(plsdStakingContract).depositASIC(_asicToTransfer) (CommunityCarnivalASICMiner.sol#665)
        - IERC20(ASIC).safeTransfer(waatcaPool,_asicToTransfer) (CommunityCarnivalASICMiner.sol#666)
        Event emitted after the call(s):
        - ReloadPeriodStart(msg.sender,currentSessionId,block.timestamp) (CommunityCarnivalASICMiner.sol#670)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

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

Low level call in Address.sendValue(address,uint256) (CommunityCarnivalASICMiner.sol#130-135):
        - (success) = recipient.call{value: amount}() (CommunityCarnivalASICMiner.sol#133)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (CommunityCarnivalASICMiner.sol#198-209):
        - (success,returndata) = target.call{value: value}(data) (CommunityCarnivalASICMiner.sol#207)
Low level call in Address.functionStaticCall(address,bytes,string) (CommunityCarnivalASICMiner.sol#227-236):
        - (success,returndata) = target.staticcall(data) (CommunityCarnivalASICMiner.sol#234)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter CarnivalCommunityASICMiner.deposit(uint256)._asicAmount (CommunityCarnivalASICMiner.sol#560) is not in mixedCase
Parameter CarnivalCommunityASICMiner.depositCARNToTrappedPool(uint256)._carnAmount (CommunityCarnivalASICMiner.sol#711) is not
 in mixedCase
Variable CarnivalCommunityASICMiner.CARN (CommunityCarnivalASICMiner.sol#498) is not in mixedCase
Variable CarnivalCommunityASICMiner.ASIC (CommunityCarnivalASICMiner.sol#499) is not in mixedCase
Variable CarnivalCommunityASICMiner.PLSB (CommunityCarnivalASICMiner.sol#500) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

CarnivalCommunityASICMiner.slitherConstructorConstantVariables() (CommunityCarnivalASICMiner.sol#488-743) uses literals with t
oo many digits:
        - TRAPPED_POOL_TARGET = 100000 * 1e12 (CommunityCarnivalASICMiner.sol#495)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
CommunityCarnivalASICMiner.sol analyzed (7 contracts with 84 detectors), 46 result(s) found
```

Slither log >> PLSDStaker.sol

```
PLSDStaker.constructor(uint256,uint256,uint256,uint256,uint256,address,address,address,address,address,address)._CARN (PLSDSta
ker.sol#544) lacks a zero-check on :
                - CARN = _CARN (PLSDStaker.sol#557)
PLSDStaker.constructor(uint256,uint256,uint256,uint256,uint256,address,address,address,address,address,address)._PLSD (PLSDSta
ker.sol#545) lacks a zero-check on :
                - PLSD = _PLSD (PLSDStaker.sol#558)
PLSDStaker.constructor(uint256,uint256,uint256,uint256,uint256,address,address,address,address,address,address)._PLSB (PLSDSta
ker.sol#546) lacks a zero-check on :
                - PLSB = _PLSB (PLSDStaker.sol#559)
PLSDStaker.constructor(uint256,uint256,uint256,uint256,uint256,address,address,address,address,address,address)._ASIC (PLSDSta
ker.sol#547) lacks a zero-check on :
                - ASIC = _ASIC (PLSDStaker.sol#560)
PLSDStaker.constructor(uint256,uint256,uint256,uint256,uint256,address,address,address,address,address,address)._HEX (PLSDStak
er.sol#548) lacks a zero-check on :
                - HEX = _HEX (PLSDStaker.sol#561)
PLSDStaker.constructor(uint256,uint256,uint256,uint256,uint256,address,address,address,address,address,address)._CARNSplitter
(PLSDStaker.sol#549) lacks a zero-check on :
                - CARNSplitter = _CARNSplitter (PLSDStaker.sol#562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

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

Low level call in Address.sendValue(address,uint256) (PLSDStaker.sol#131-136):
        - (success) = recipient.call{value: amount}() (PLSDStaker.sol#134)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (PLSDStaker.sol#199-210):
        - (success,returndata) = target.call{value: value}(data) (PLSDStaker.sol#208)
Low level call in Address.functionStaticCall(address,bytes,string) (PLSDStaker.sol#228-237):
        - (success,returndata) = target.staticcall(data) (PLSDStaker.sol#235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter PLSDStaker.stake(uint256)._amount (PLSDStaker.sol#566) is not in mixedCase
Parameter PLSDStaker.depositPLSD(uint256)._amount (PLSDStaker.sol#686) is not in mixedCase
Parameter PLSDStaker.depositPLSB(uint256)._amount (PLSDStaker.sol#699) is not in mixedCase
Parameter PLSDStaker.depositASIC(uint256)._amount (PLSDStaker.sol#712) is not in mixedCase
Parameter PLSDStaker.depositHEX(uint256)._amount (PLSDStaker.sol#726) is not in mixedCase
Variable PLSDStaker.CARN (PLSDStaker.sol#461) is not in mixedCase
Variable PLSDStaker.PLSD (PLSDStaker.sol#462) is not in mixedCase
Variable PLSDStaker.PLSB (PLSDStaker.sol#463) is not in mixedCase
Variable PLSDStaker.ASIC (PLSDStaker.sol#464) is not in mixedCase
Variable PLSDStaker.HEX (PLSDStaker.sol#465) is not in mixedCase
Variable PLSDStaker.CARNSplitter (PLSDStaker.sol#467) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Variable PLSDStaker.claimRewards()._plsbReward (PLSDStaker.sol#764) is too similar to PLSDStaker.claimRewards()._plsdReward (P
LSDStaker.sol#760)
Variable PLSDStaker.plsbPendingRewards (PLSDStaker.sol#450) is too similar to PLSDStaker.plsdPendingRewards (PLSDStaker.sol#44
6)
Variable PLSDStaker.plsbRewardPool (PLSDStaker.sol#449) is too similar to PLSDStaker.plsdRewardPool (PLSDStaker.sol#445)
Variable PLSDStaker.plsbRewardPoolTotal (PLSDStaker.sol#451) is too similar to PLSDStaker.plsdRewardPoolTotal (PLSDStaker.sol#
447)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
PLSDStaker.sol analyzed (5 contracts with 84 detectors), 40 result(s) found
```

Slither log >> PulseBitcoinLockRewards.sol

```
PulseBitcoinLockNFTRewards.constructor(address,address)._waatcaAddress (PulseBitcoinLockRewards.sol#164) lacks a zero-check on
 :
                - waatcaAddress = _waatcaAddress (PulseBitcoinLockRewards.sol#165)
PulseBitcoinLockNFTRewards.constructor(address,address)._pulseBitcoinLockNftContractAddress (PulseBitcoinLockRewards.sol#164)
lacks a zero-check on :
                - pulseBitcoinLockNftContractAddress = _pulseBitcoinLockNftContractAddress (PulseBitcoinLockRewards.sol#166)
PulseBitcoinLockNFTRewards.setCarnAddress(address)._rewardTokenCARN (PulseBitcoinLockRewards.sol#170) lacks a zero-check on :
                - CARN = _rewardTokenCARN (PulseBitcoinLockRewards.sol#171)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in PulseBitcoinLockNFTRewards.registerNftForRewards(uint256) (PulseBitcoinLockRewards.sol#211-264):
        External calls:
        - IERC20(CARN).transfer(msg.sender,tokenIdsToDailyRewardAmount[tokenId] * 100) (PulseBitcoinLockRewards.sol#255)
        - IERC20(CARN).transfer(waatcaAddress,tokenIdsToDailyRewardAmount[tokenId] * 100) (PulseBitcoinLockRewards.sol#256)
        - IERC20(CARN).transfer(msg.sender,tokenIdsToDailyRewardAmount[tokenId]) (PulseBitcoinLockRewards.sol#258)
        - IERC20(CARN).transfer(waatcaAddress,tokenIdsToDailyRewardAmount[tokenId]) (PulseBitcoinLockRewards.sol#259)
        State variables written after the call(s):
        - tokenIdsToEndRewardsDay[tokenId] = _currentDay() + numDaysLockedUpFromRegistration (PulseBitcoinLockRewards.sol#263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

PulseBitcoinLockNFTRewards.withdrawRewards(uint256) (PulseBitcoinLockRewards.sol#174-209) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(tokenIdsToLastWithdrawalDay[tokenId] < tokenIdsToEndRewardsDay[tokenId],You have already recieve
ed all possible rewards for this NFT) (PulseBitcoinLockRewards.sol#183-187)
        - require(bool,string)(_currentDay() > tokenIdsToLastWithdrawalDay[tokenId],Cannot withdraw twice on the same day, try
 again tomorrow) (PulseBitcoinLockRewards.sol#188-191)
        - numOfDaysSinceLastWithdrawal > totalDaysOfRewardsLeft (PulseBitcoinLockRewards.sol#197)
PulseBitcoinLockNFTRewards.registerNftForRewards(uint256) (PulseBitcoinLockRewards.sol#211-264) uses timestamp for comparisons
        Dangerous comparisons:
        - numDaysLockedUpFromRegistration > 1000 (PulseBitcoinLockRewards.sol#227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Context._msgData() (PulseBitcoinLockRewards.sol#94-97) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

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

Parameter PulseBitcoinLockNFTRewards.setCarnAddress(address)._rewardTokenCARN (PulseBitcoinLockRewards.sol#170) is not in mixe
dCase
Variable PulseBitcoinLockNFTRewards.CARN (PulseBitcoinLockRewards.sol#154) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (PulseBitcoinLockRewards.sol#95)" inContext (PulseBitcoinLockRewards.sol#89-98)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

PulseBitcoinLockNFTRewards.pulseBitcoinLockNftContract (PulseBitcoinLockRewards.sol#156) should be immutable
PulseBitcoinLockNFTRewards.pulseBitcoinLockNftContractAddress (PulseBitcoinLockRewards.sol#157) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
PulseBitcoinLockRewards.sol analyzed (5 contracts with 84 detectors), 21 result(s) found
```

Slither log >> WaatcaNFT.sol

```
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._name (WaatcaNFT.sol#1214
) shadows:
        - ERC721._name (WaatcaNFT.sol#537) (state variable)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._symbol (WaatcaNFT.sol#12
15) shadows:
        - ERC721._symbol (WaatcaNFT.sol#540) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._purchaseTokenUSDC (Waatc
aNFT.sol#1206) lacks a zero-check on :
                - purchaseTokenUSDC = _purchaseTokenUSDC (WaatcaNFT.sol#1219)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._rewardTokenPLSD (WaatcaN
FT.sol#1207) lacks a zero-check on :
                - rewardTokenPLSD = _rewardTokenPLSD (WaatcaNFT.sol#1220)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._rewardTokenPLSB (WaatcaN
FT.sol#1208) lacks a zero-check on :
                - rewardTokenPLSB = _rewardTokenPLSB (WaatcaNFT.sol#1221)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._rewardTokenASIC (WaatcaN
FT.sol#1209) lacks a zero-check on :
                - rewardTokenASIC = _rewardTokenASIC (WaatcaNFT.sol#1222)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._rewardTokenHEX (WaatcaNF
T.sol#1210) lacks a zero-check on :
                - rewardTokenHEX = _rewardTokenHEX (WaatcaNFT.sol#1223)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._rewardTokenUSDC (WaatcaN
FT.sol#1211) lacks a zero-check on :
                - rewardTokenUSDC = _rewardTokenUSDC (WaatcaNFT.sol#1224)
WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._carnivalAddress (WaatcaN
FT.sol#1212) lacks a zero-check on :
                - carnivalAddress = _carnivalAddress (WaatcaNFT.sol#1225)
WaatcaNFT.setCarnAddress(address)._rewardTokenCARN (WaatcaNFT.sol#1228) lacks a zero-check on :
                - rewardTokenCARN = _rewardTokenCARN (WaatcaNFT.sol#1229)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
Variable WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,string,string)._rewardTokenPLSB
 (WaatcaNFT.sol#1208) is too similar to WaatcaNFT.constructor(address,address,address,address,address,address,address,uint256,
string,string)._rewardTokenPLSD (WaatcaNFT.sol#1207)
Variable WaatcaNFT.rewardTokenPLSB (WaatcaNFT.sol#1170) is too similar to WaatcaNFT.rewardTokenPLSD (WaatcaNFT.sol#1169)
Variable WaatcaNFT._burn(uint256).withdrawablePortionOfASIC (WaatcaNFT.sol#1293) is too similar to WaatcaNFT._burn(uint256).wi
thdrawablePortionOfUSDC (WaatcaNFT.sol#1296)
Variable WaatcaNFT._burn(uint256).withdrawablePortionOfPLSB (WaatcaNFT.sol#1292) is too similar to WaatcaNFT._burn(uint256).wi
thdrawablePortionOfPLSD (WaatcaNFT.sol#1291)
Variable WaatcaNFT._burn(uint256).withdrawablePortionOfPLSD (WaatcaNFT.sol#1291) is too similar to WaatcaNFT._burn(uint256).wi
thdrawablePortionOfUSDC (WaatcaNFT.sol#1296)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

WaatcaNFT.mintDeadline (WaatcaNFT.sol#1187) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
WaatcaNFT.sol analyzed (15 contracts with 84 detectors), 60 result(s) found
```

Solidity Static Analysis

CarnTokenBooth.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: 36:19:

Gas & Economy

Gas costs:

Gas requirement of function CarnTokenBooth.pulseBitcoinLockNFTRewardPoolAddress 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: 18:4:

Miscellaneous

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 58:8:

CommunityCarnivalASICMiner.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: 313:40:

Gas & Economy

Gas costs:

Gas requirement of function CarnivalCommunityASICMiner.depositCARNToTrappedPool 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: 283:4:

Miscellaneous

No return:

PulseBitcoin.minerList(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 41:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 310:8:

PLSDStaker.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: 319:16:

Gas & Economy

Gas costs:

Gas requirement of function PLSDStaker.claimRewards 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: 313:4:

Miscellaneous

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 329:12:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 247:8:

PulseBitcoinLockRewards.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in PulseBitcoinLockNFTRewards.registerNftForRewards(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 73:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 133:16:

Gas & Economy

Gas costs:

Gas requirement of function PulseBitcoinLockNFTRewards.currentDay 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: 128:4:

Miscellaneous

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 78:8:

WaatcaNFT.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WaatcaNFT._burn(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 130:4:

Gas & Economy

Gas costs:

Gas requirement of function WaatcaNFT.purchaseTokenUSDC 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: 18:4:

Miscellaneous

Solhint Linter

CarnTokenBooth.sol

```
Carnival.sol:2:1: Error: Compiler version ^0.8.16 does not satisfy
the r semver requirement
Carnival.sol:16:5: Error: Explicitly mark visibility of state
Carnival.sol:21:30: Error: Variable name must be in mixedCase
Carnival.sol:23:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Carnival.sol:27:9: Error: Variable name must be in
mixedCaseCarnival.sol:40:13: Error: Avoid to make time-based
decisions in your business logic
```

CommunityCarnivalASICMiner.sol

```
CommunityCarnivalASICMiner.sol:246:13: Error: Avoid to make
time-based decisions in your business logic
CommunityCarnivalASICMiner.sol:274:13: Error: Avoid to make
time-based decisions in your business logic
CommunityCarnivalASICMiner.sol:286:64: Error: Avoid to make
time-based decisions in your business logic
CommunityCarnivalASICMiner.sol:306:41: Error: Avoid to make
time-based decisions in your business logic
CommunityCarnivalASICMiner.sol:313:41: Error: Avoid to make
time-based decisions in your business logic
```

PLSDStaker.sol

```
PLSDStaker.sol:2:1: Error: Compiler version ^0.8.16 does not satisfy
the r semver requirement
PLSDStaker.sol:8:1: Error: Contract has 19 states declarations but
allowed no more than 15
PLSDStaker.sol:40:30: Error: Variable name must be in mixedCase
PLSDStaker.sol:111:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
PLSDStaker.sol:121:9: Error: Variable name must be in mixedCase
PLSDStaker.sol:122:9: Error: Variable name must be in mixedCase
PLSDStaker.sol:226:24: Error: Avoid to make time-based decisions in
your business logic
PLSDStaker.sol:231:63: Error: Avoid to make time-based decisions in
your business logic
PLSDStaker.sol:314:13: Error: Avoid to make time-based decisions in
your business logic
PLSDStaker.sol:319:17: Error: Avoid to make time-based decisions in
your business logic
```

PulseBitcoinLockRewards.sol

```
PulseBitcoinLockRewards.sol:2:1: Error: Compiler version ^0.8.0 does
not satisfy the r semver requirement
PulseBitcoinLockRewards.sol:16:20: Error: Variable name must be in
mixedCase
PulseBitcoinLockRewards.sol:18:5: Error: Explicitly mark visibility
of state
PulseBitcoinLockRewards.sol:26:5: Error: Explicitly mark visibility
in function (Set ignoreConstructors to true if using solidity
>=0.7.0)
PulseBitcoinLockRewards.sol:70:9: Error: Possible reentrancy
vulnerabilities. Avoid state changes after transfer.
PulseBitcoinLockRewards.sol:125:9: Error: Possible reentrancy
vulnerabilities. Avoid state changes after transfer.
PulseBitcoinLockRewards.sol:133:17: Error: Avoid to make time-based
decisions in your business logic
```

WaatcaNFT.sol

```
WaatcaNFT.sol:2:1: Error: Compiler version ^0.8.0 does not satisfy
the r semver requirement
WaatcaNFT.sol:57:5: Error: Explicitly mark visibility in
WaatcaNFT.sol:164:9: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
WaatcaNFT.sol:165:9: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
WaatcaNFT.sol:171:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
WaatcaNFT.sol:171:30: 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