EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Pulse Rich
Domain:	dev.pulserich.app
Platform:	Ethereum PLUS Network
Language:	Solidity
Date:	January 19th, 2024

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.

Introduction

EtherAuthority was contracted by the Pulse Rich team to perform the Security audit of the Pulse Rich 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 19th, 2024.

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

- Pulse Rich is a contract that can be divided into multiples, each with unique functionalities:
  - **PulseRichards:** This contract is utilized for mint NFTs, which can be paid with USDC, HEX, eHEX, PLSX, or PLS tokens.
  - **GenesisPLSRewards:** This contract is utilized for withdrawing rewards and registering a new Nft for rewards.

- There are 2 smart contracts, which were included in the audit scope.
- The Pulse Rich NFT contract inherits Strings, SafeERC20, IERC20, ReentrancyGuard standard smart contracts from the OpenZeppelin library. An ERC721r contract inherited from the middlemarch contracts.These OpenZeppelin contracts and middlemarch contracts are considered community audited and time tested, and hence are not part of the audit scope.
- The token is without any other custom functionality and without any ownership control, which makes it truly decentralized.

Audit scope

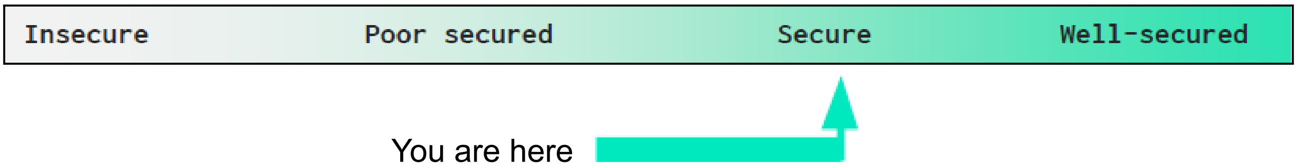
Name	**Code Review and Security Analysis Report for Pulse Rich Smart Contracts**
Platform	**Ethereum PLUS Network**
Language	**Solidity**
File 1	genesis.sol
File 1 MD5 Hash	957AAC921E9371F9DF86CF824C33A540
File 2	GenesisPLSRewards.sol
File 2 MD5 Hash	1A6B5DA921A7C55E46C3459CF4CBE39C
Audit Date	January 19th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 PulseRichards.sol ● This contract is utilized for mint NFTs, which can be paid with USDC, HEX, eHEX, PLSX, or PLS tokens. ● OpenZeppelin library used. ● Middlemarch library used. **Ownership Control:** ● There are no owner functions, which makes it 100% decentralized.	YES, This is valid.
File 2 GenesisPLSRewards.sol ● The owner of NFT can withdraw rewards. ● The owner of NFT can register a new NFT for rewards. ● OpenZeppelin library used. **Ownership Control:** ● There are no owner functions, which makes it 100% decentralized.	YES, This is valid.

Audit Summary

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



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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, 0 low and 4 very low level issues.

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	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	Moderated
	Features claimed	Passed
	Other programming issues	Moderated
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 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Pulse Rich 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 Pulse Rich Protocol.

The Pulse Rich 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 well commented on smart contracts.

Documentation

We were given a Pulse Rich smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are well commented on. And 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.

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

genesis.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	usdcMint	external	non Reentrant	No Issue
3	eHexMint	external	non Reentrant	No Issue
4	hexMint	external	non Reentrant	No Issue
5	plsxMint	external	non Reentrant	No Issue
6	plsMint	external	non Reentrant	No Issue
7	tokenURI	read	Passed	No Issue
8	getTokenIdsByWallet	external	Passed	No Issue
9	totalSupply	read	Passed	No Issue
10	name	read	Passed	No Issue
11	symbol	read	Passed	No Issue
12	numberMinted	read	Passed	No Issue
13	_mintRandom	internal	Passed	No Issue
14	_mintAtIndex	internal	Passed	No Issue
15	getAvailableTokenAtIndex	write	Passed	No Issue
16	_setExtraAddressData	internal	Passed	No Issue
17	_getAddressExtraData	internal	Passed	No Issue
18	_incrementAmountMinted	write	Passed	No Issue
19	nonReentrant	modifier	Passed	No Issue
20	_nonReentrantBefore	write	Passed	No Issue
21	_nonReentrantAfter	write	Passed	No Issue
22	_reentrancyGuardEntered	internal	Passed	No Issue

GenesisPLSRewards.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	receive	external	Passed	No Issue
3	withdrawRewards	write	Passed	No Issue
4	registerNftForRewards	write	Passed	No Issue
5	currentDay	external	Passed	No Issue
6	_currentDay	internal	Passed	No Issue
7	_senderIsTokenOwner	internal	Passed	No Issue
8	bulkRegister	write	non Reentrant	No Issue
9	bulkWithdraw	write	‘non Reentrant	No Issue
10	nonReentrant	modifier	Passed	No Issue
11	_nonReentrantBefore	write	Passed	No Issue
12	_nonReentrantAfter	write	Passed	No Issue
13	_reentrancyGuardEntered	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.

Audit Findings

Critical Severity

No critical severity vulnerabilities were found in the contract code.

High Severity

No high severity vulnerabilities were found in the contract code.

Medium

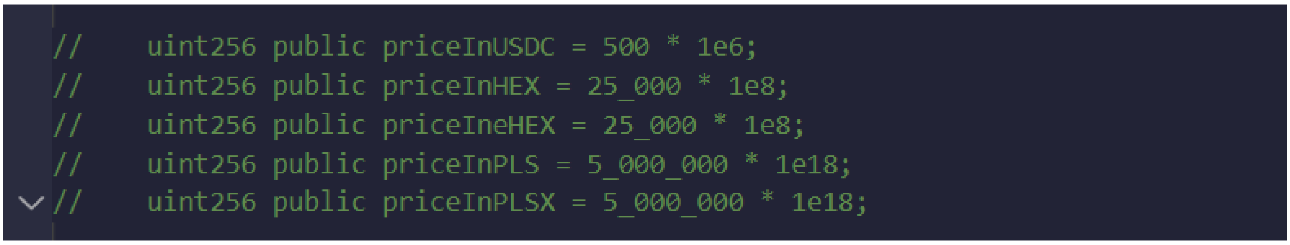
No medium severity vulnerabilities were found in the contract code.

Low

No low severity vulnerabilities were found in the contract code.

Very Low / Informational / Best practices:

(1) Commented code: **genesis.sol**



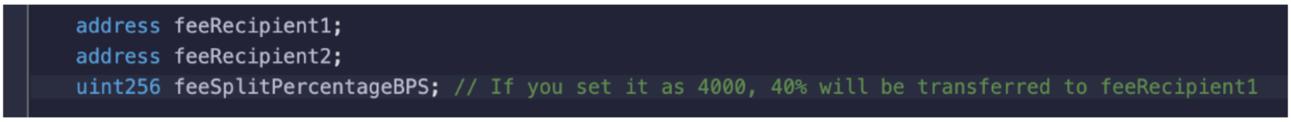
```
//    uint256 public priceInUSDC = 500 * 1e6;
//    uint256 public priceInHEX = 25_000 * 1e8;
//    uint256 public priceIneHEX = 25_000 * 1e8;
//    uint256 public priceInPLS = 5_000_000 * 1e18;
//    uint256 public priceInPLSX = 5_000_000 * 1e18;
```

Commented code is present.

Resolution: Please remove commented code as code of standard.

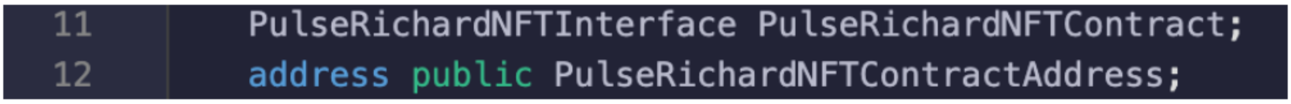
(2) Parameter can be immutable:

genesis.sol



```
    address feeRecipient1;
    address feeRecipient2;
    uint256 feeSplitPercentageBPS; // If you set it as 4000, 40% will be transferred to feeRecipient1
```

GenesisPLSRewards.sol

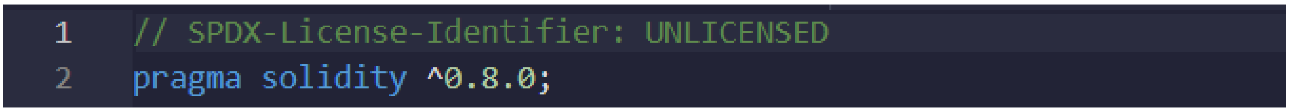


```
    PulseRichardNFTInterface PulseRichardNFTContract;
    address public PulseRichardNFTContractAddress;
```

Variables that are set within the constructor but further remain unchanged should be marked as immutable to save gas and to ease the reviewing process of third-parties.

Resolution: Consider marking this variable as immutable.

(3) Use latest solidity version: **GenesisPLSRewards.sol**

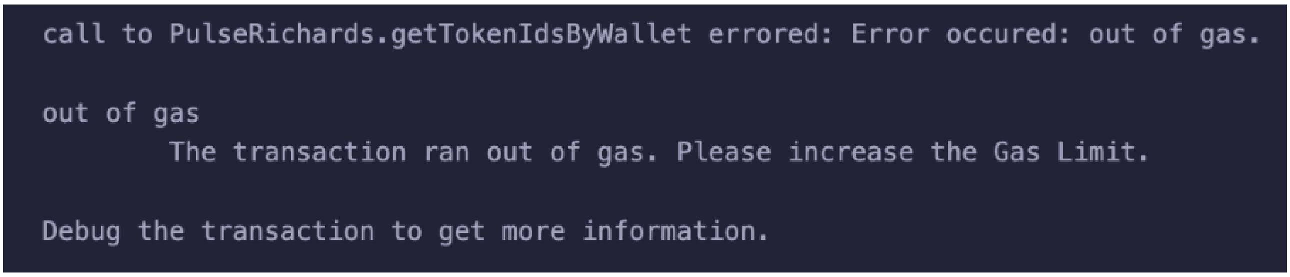


```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use the latest solidity versions.

(4) Not able to get token ID for wallet address: **genesis.sol**



```
call to PulseRichards.getTokenIdsByWallet errored: Error occured: out of gas.

out of gas
        The transaction ran out of gas. Please increase the Gas Limit.

Debug the transaction to get more information.
```

Since the NFT's are minted in random so token ids are in random so to view token ids for particular wallet the to and from token ids will be in large numbers so iterating large numbers will result in out of gas error.

Centralization Risk

The Pulse Rich smart contract does not have any ownership control, hence it is 100% decentralized.

Therefore, there is **no** centralization risk.

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on given objects as files. We had observed 4 Informational severity issues in the smart contracts. but those are not critical. **So, the smart contracts are ready 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.

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

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

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of 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.

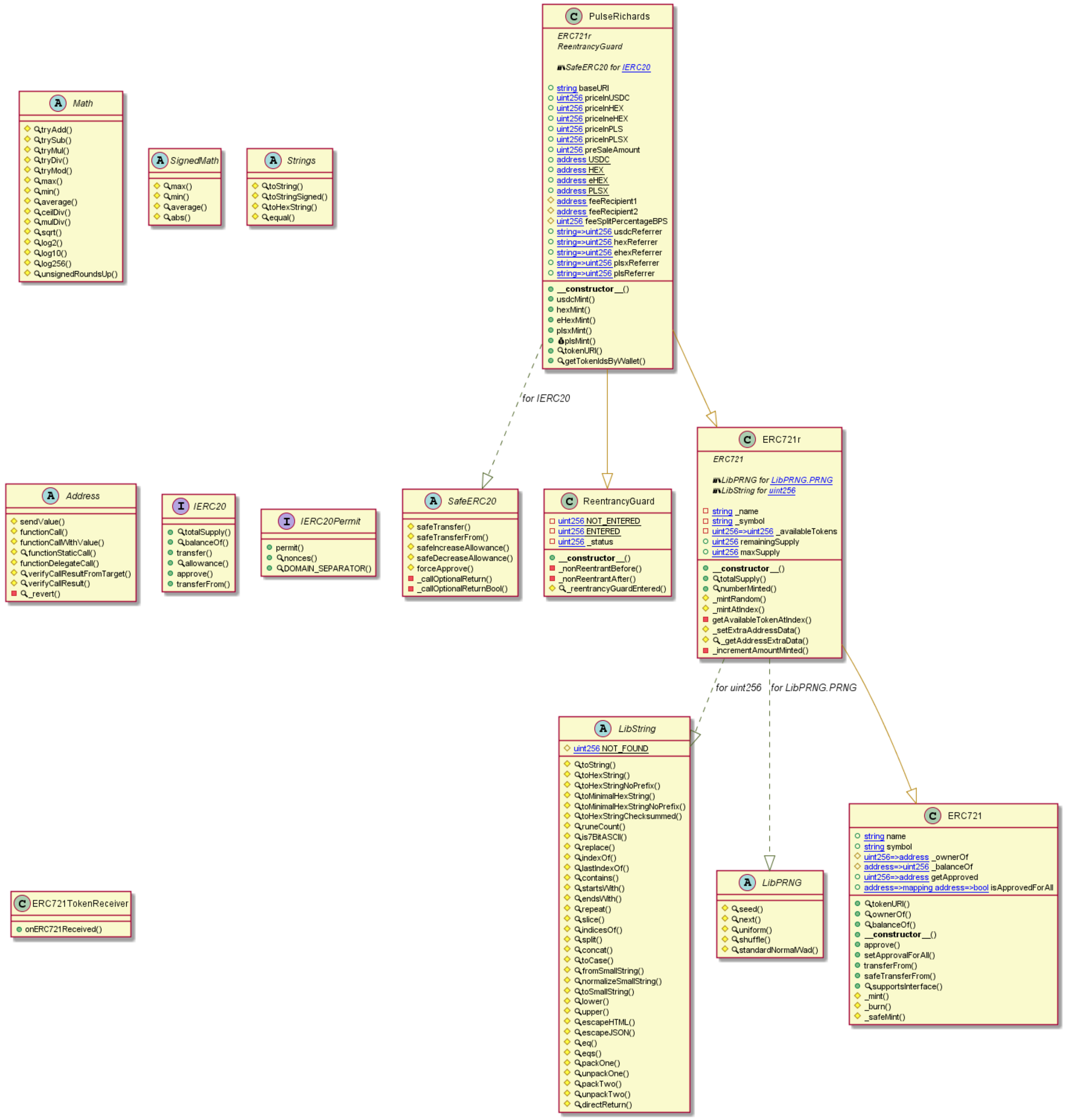
Technical Disclaimer

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.

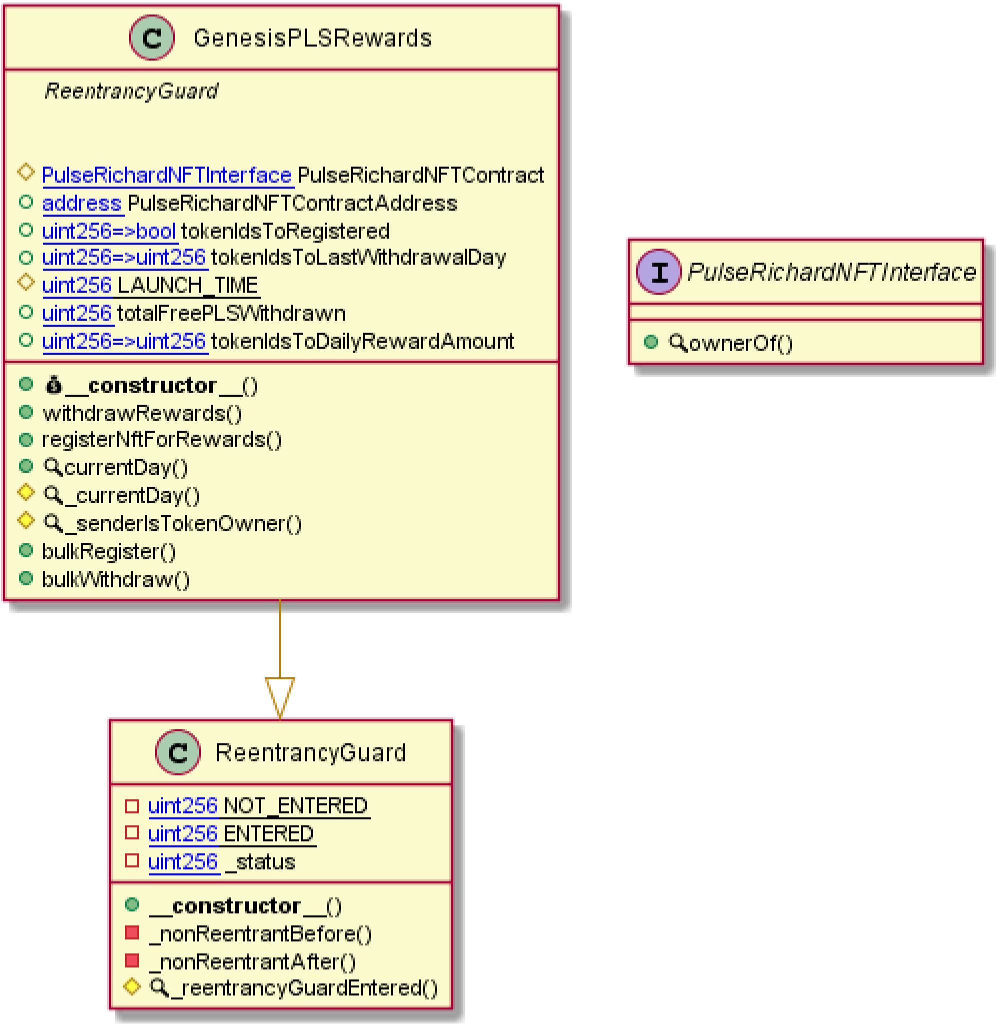
Appendix

Code Flow Diagram - Pulse Rich Protocol

genesis Diagram



GenesisPLSRewards Diagram

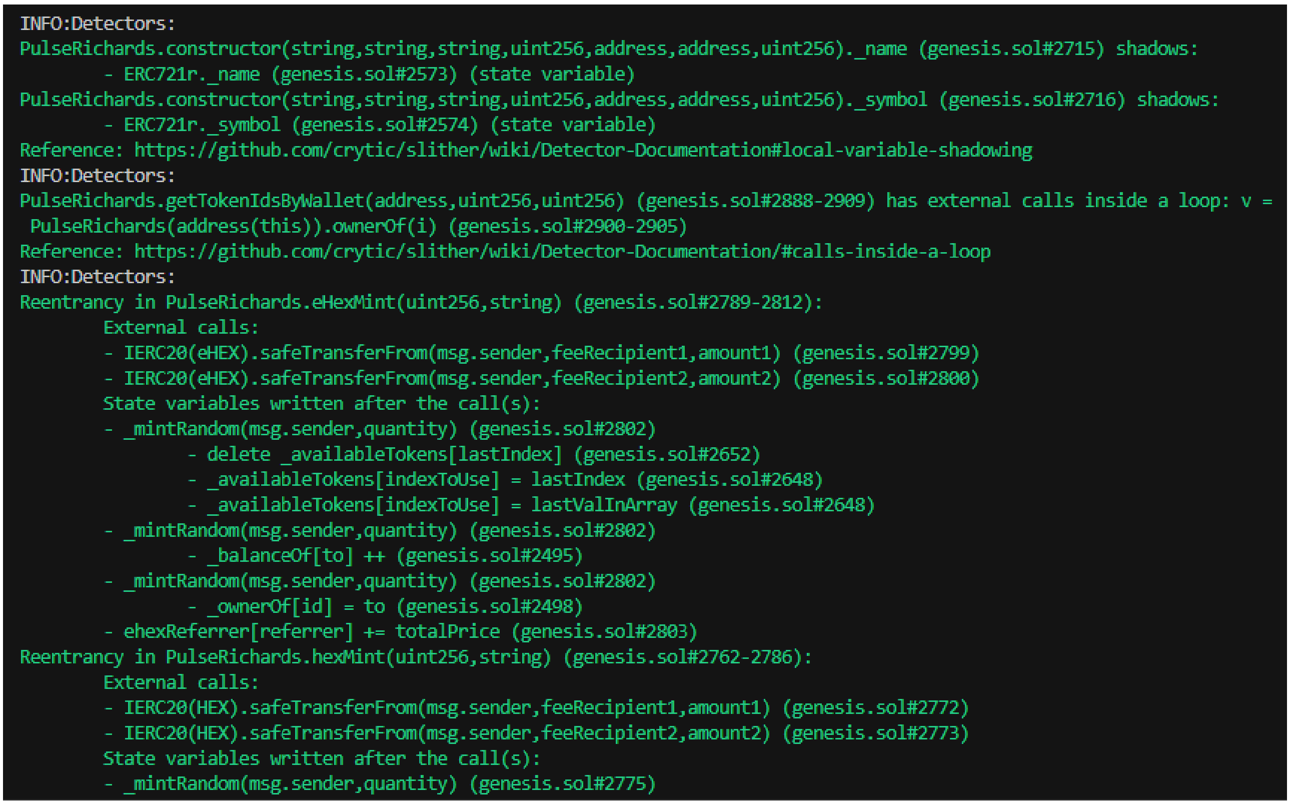


Slither Results Log

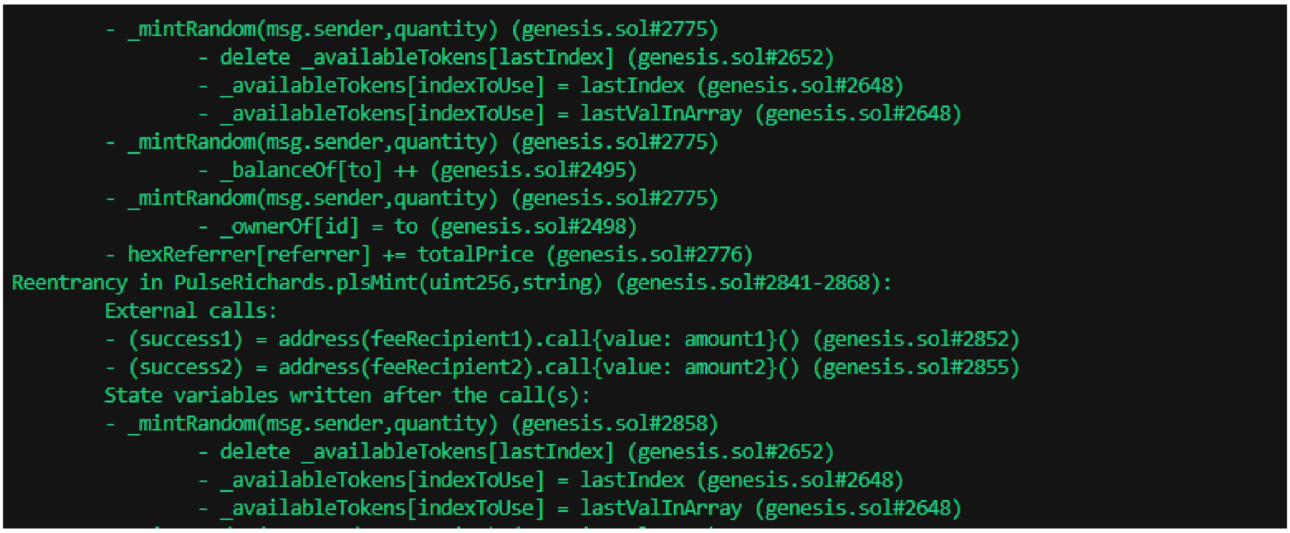
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

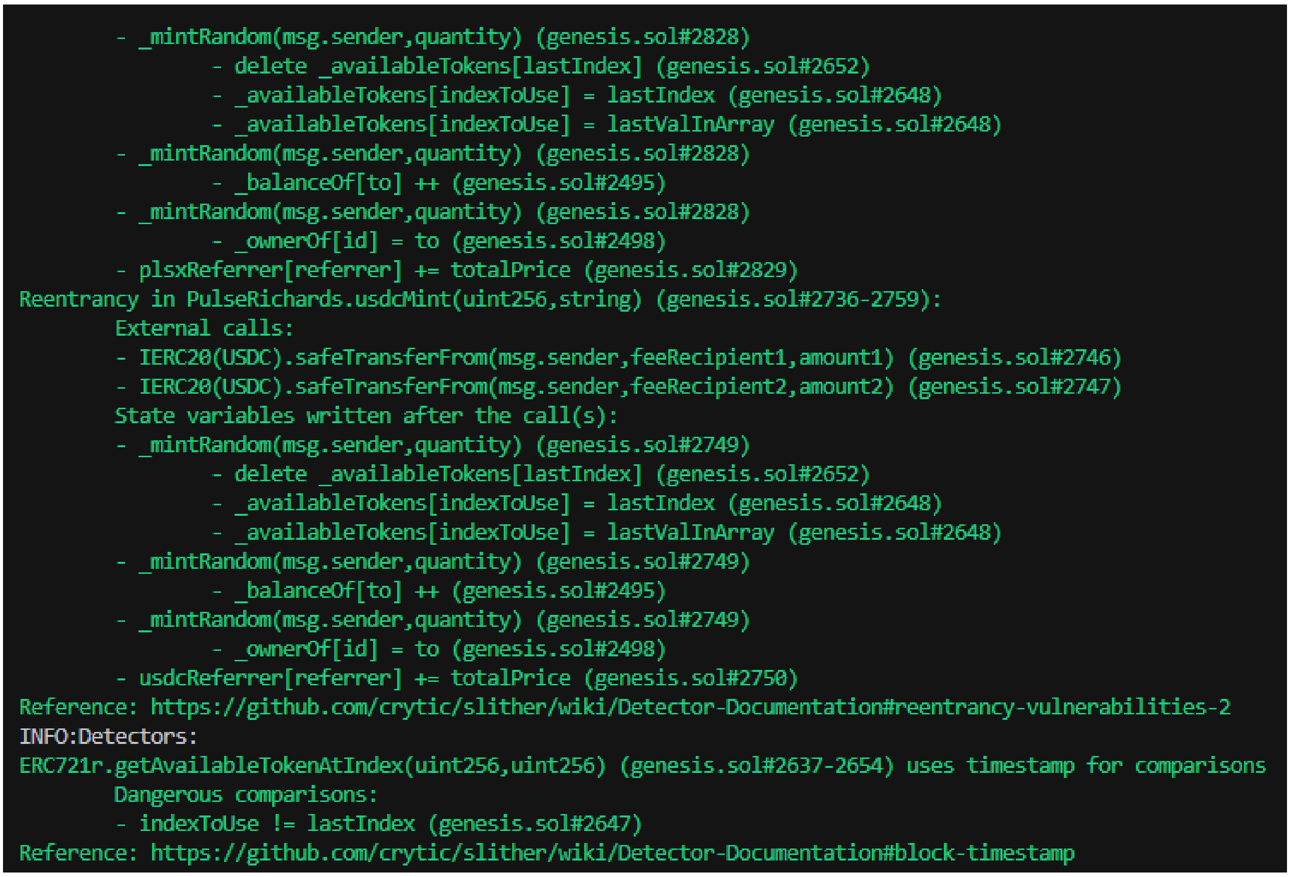
Slither log >> PulseRichards.sol



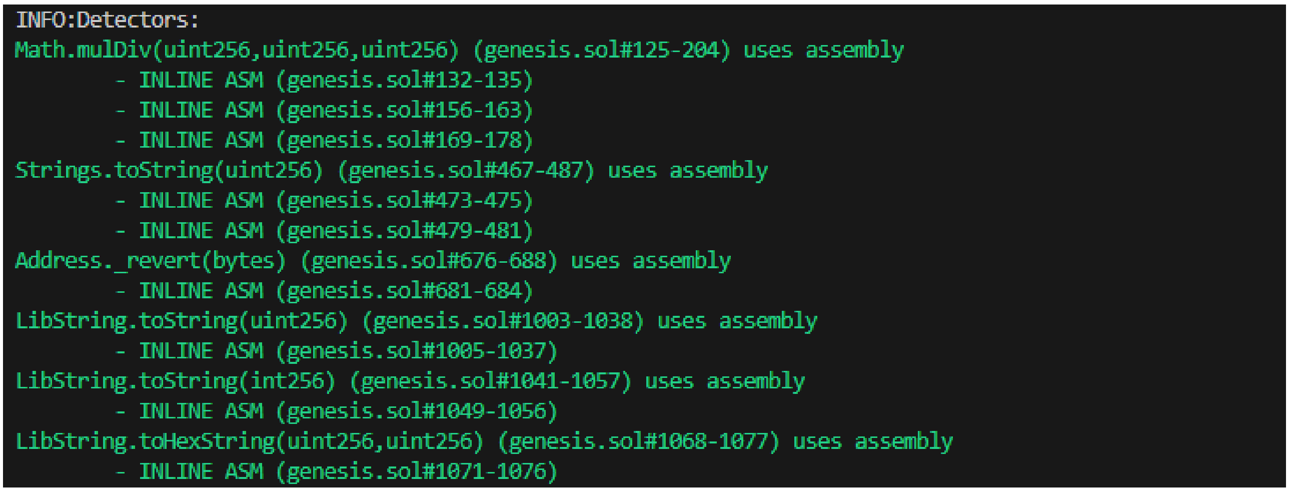
```
INFO:Detectors:
PulseRichards.constructor(string,string,string,uint256,address,address,uint256)._name (genesis.sol#2715) shadows:
        - ERC721r._name (genesis.sol#2573) (state variable)
PulseRichards.constructor(string,string,string,uint256,address,address,uint256)._symbol (genesis.sol#2716) shadows:
        - ERC721r._symbol (genesis.sol#2574) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
PulseRichards.getTokenIdsByWallet(address,uint256,uint256) (genesis.sol#2888-2909) has external calls inside a loop: v =
 PulseRichards(address(this)).ownerOf(i) (genesis.sol#2900-2905)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in PulseRichards.eHexMint(uint256,string) (genesis.sol#2789-2812):
        External calls:
        - IERC20(eHEX).safeTransferFrom(msg.sender,feeRecipient1,amount1) (genesis.sol#2799)
        - IERC20(eHEX).safeTransferFrom(msg.sender,feeRecipient2,amount2) (genesis.sol#2800)
        State variables written after the call(s):
        - _mintRandom(msg.sender,quantity) (genesis.sol#2802)
                - delete _availableTokens[lastIndex] (genesis.sol#2652)
                - _availableTokens[indexToUse] = lastIndex (genesis.sol#2648)
                - _availableTokens[indexToUse] = lastValInArray (genesis.sol#2648)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2802)
                - _balanceOf[to] ++ (genesis.sol#2495)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2802)
                - _ownerOf[id] = to (genesis.sol#2498)
        - ehexReferrer[referrer] += totalPrice (genesis.sol#2803)
Reentrancy in PulseRichards.hexMint(uint256,string) (genesis.sol#2762-2786):
        External calls:
        - IERC20(HEX).safeTransferFrom(msg.sender,feeRecipient1,amount1) (genesis.sol#2772)
        - IERC20(HEX).safeTransferFrom(msg.sender,feeRecipient2,amount2) (genesis.sol#2773)
        State variables written after the call(s):
        - _mintRandom(msg.sender,quantity) (genesis.sol#2775)
```



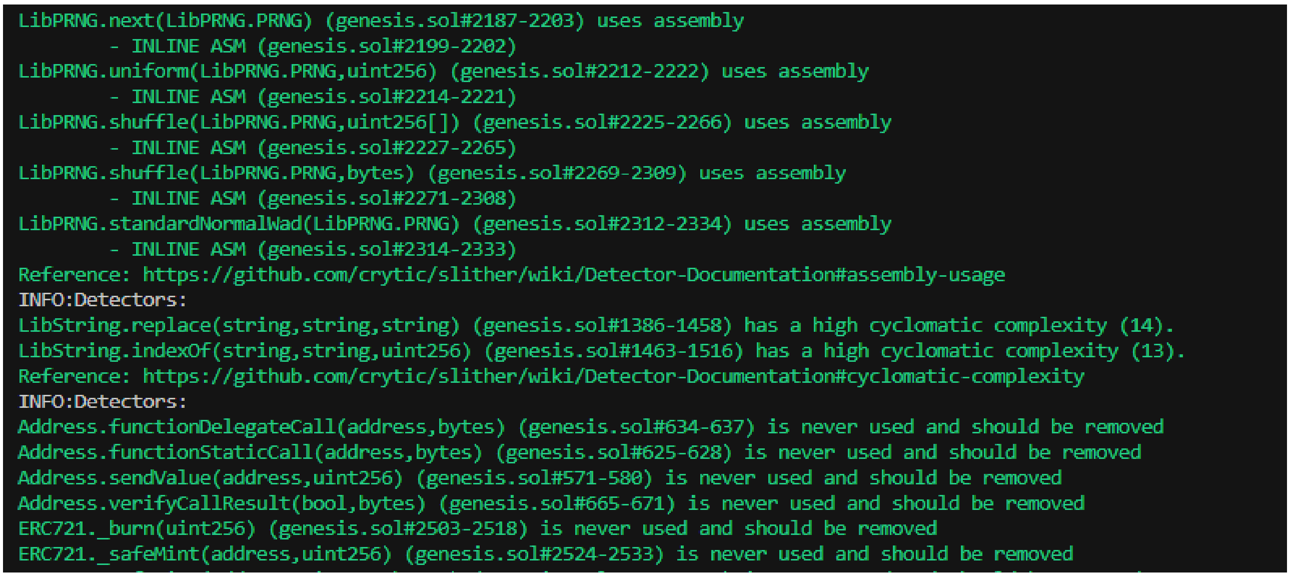
```
        - _mintRandom(msg.sender,quantity) (genesis.sol#2775)
                - delete _availableTokens[lastIndex] (genesis.sol#2652)
                - _availableTokens[indexToUse] = lastIndex (genesis.sol#2648)
                - _availableTokens[indexToUse] = lastValInArray (genesis.sol#2648)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2775)
                - _balanceOf[to] ++ (genesis.sol#2495)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2775)
                - _ownerOf[id] = to (genesis.sol#2498)
        - hexReferrer[referrer] += totalPrice (genesis.sol#2776)
Reentrancy in PulseRichards.plsMint(uint256,string) (genesis.sol#2841-2868):
        External calls:
        - (success1) = address(feeRecipient1).call{value: amount1}() (genesis.sol#2852)
        - (success2) = address(feeRecipient2).call{value: amount2}() (genesis.sol#2855)
        State variables written after the call(s):
        - _mintRandom(msg.sender,quantity) (genesis.sol#2858)
                - delete _availableTokens[lastIndex] (genesis.sol#2652)
                - _availableTokens[indexToUse] = lastIndex (genesis.sol#2648)
                - _availableTokens[indexToUse] = lastValInArray (genesis.sol#2648)
```



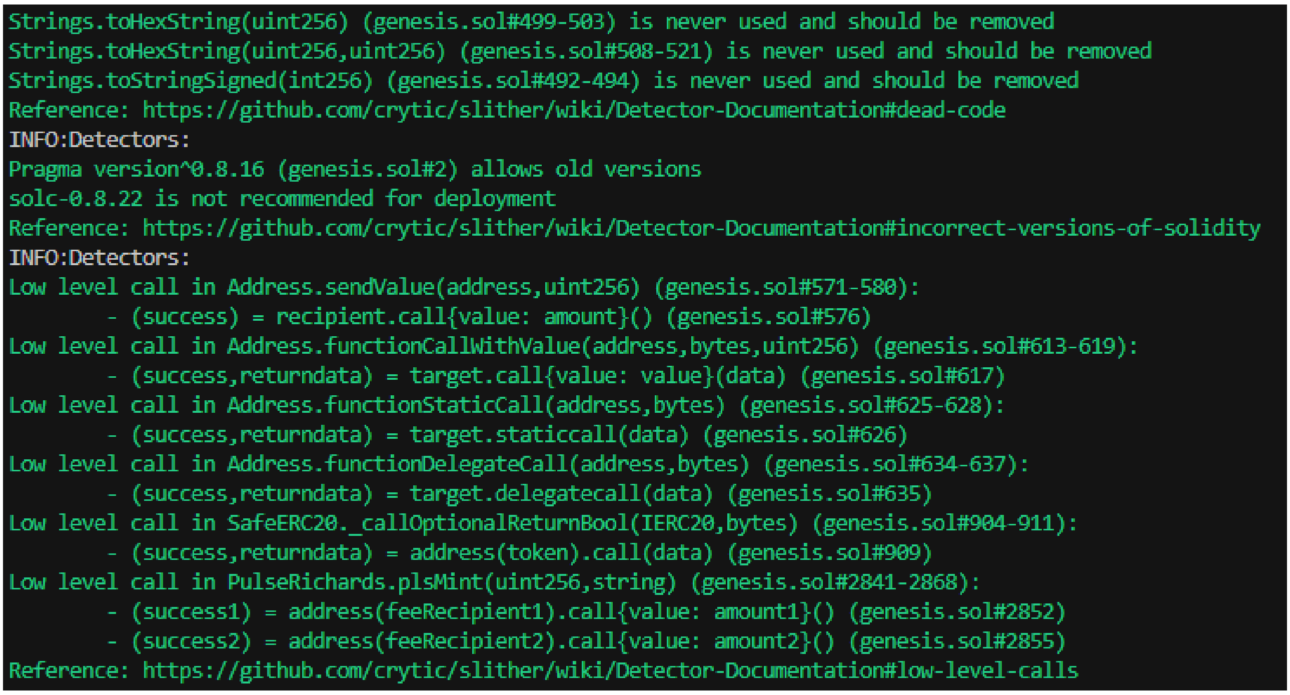
```
        - _mintRandom(msg.sender,quantity) (genesis.sol#2828)
                - delete _availableTokens[lastIndex] (genesis.sol#2652)
                - _availableTokens[indexToUse] = lastIndex (genesis.sol#2648)
                - _availableTokens[indexToUse] = lastValInArray (genesis.sol#2648)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2828)
                - _balanceOf[to] ++ (genesis.sol#2495)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2828)
                - _ownerOf[id] = to (genesis.sol#2498)
        - plsxReferrer[referrer] += totalPrice (genesis.sol#2829)
Reentrancy in PulseRichards.usdcMint(uint256,string) (genesis.sol#2736-2759):
        External calls:
        - IERC20(USDC).safeTransferFrom(msg.sender,feeRecipient1,amount1) (genesis.sol#2746)
        - IERC20(USDC).safeTransferFrom(msg.sender,feeRecipient2,amount2) (genesis.sol#2747)
        State variables written after the call(s):
        - _mintRandom(msg.sender,quantity) (genesis.sol#2749)
                - delete _availableTokens[lastIndex] (genesis.sol#2652)
                - _availableTokens[indexToUse] = lastIndex (genesis.sol#2648)
                - _availableTokens[indexToUse] = lastValInArray (genesis.sol#2648)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2749)
                - _balanceOf[to] ++ (genesis.sol#2495)
        - _mintRandom(msg.sender,quantity) (genesis.sol#2749)
                - _ownerOf[id] = to (genesis.sol#2498)
        - usdcReferrer[referrer] += totalPrice (genesis.sol#2750)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
ERC721r.getAvailableTokenAtIndex(uint256,uint256) (genesis.sol#2637-2654) uses timestamp for comparisons
        Dangerous comparisons:
        - indexToUse != lastIndex (genesis.sol#2647)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



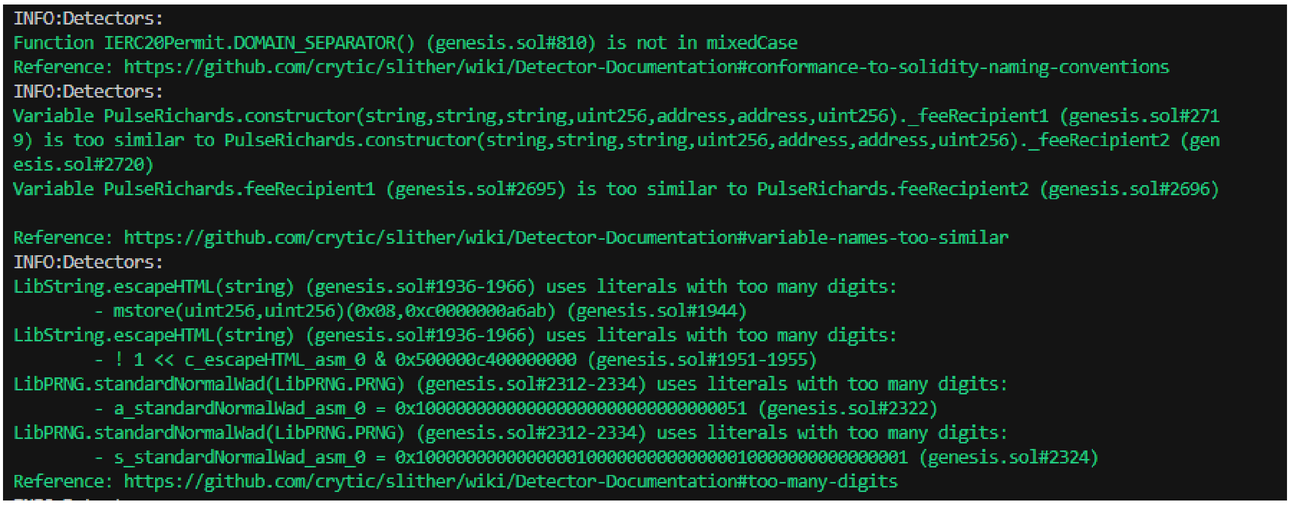
```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (genesis.sol#125-204) uses assembly
        - INLINE ASM (genesis.sol#132-135)
        - INLINE ASM (genesis.sol#156-163)
        - INLINE ASM (genesis.sol#169-178)
Strings.toString(uint256) (genesis.sol#467-487) uses assembly
        - INLINE ASM (genesis.sol#473-475)
        - INLINE ASM (genesis.sol#479-481)
Address._revert(bytes) (genesis.sol#676-688) uses assembly
        - INLINE ASM (genesis.sol#681-684)
LibString.toString(uint256) (genesis.sol#1003-1038) uses assembly
        - INLINE ASM (genesis.sol#1005-1037)
LibString.toString(int256) (genesis.sol#1041-1057) uses assembly
        - INLINE ASM (genesis.sol#1049-1056)
LibString.toHexString(uint256,uint256) (genesis.sol#1068-1077) uses assembly
        - INLINE ASM (genesis.sol#1071-1076)
```



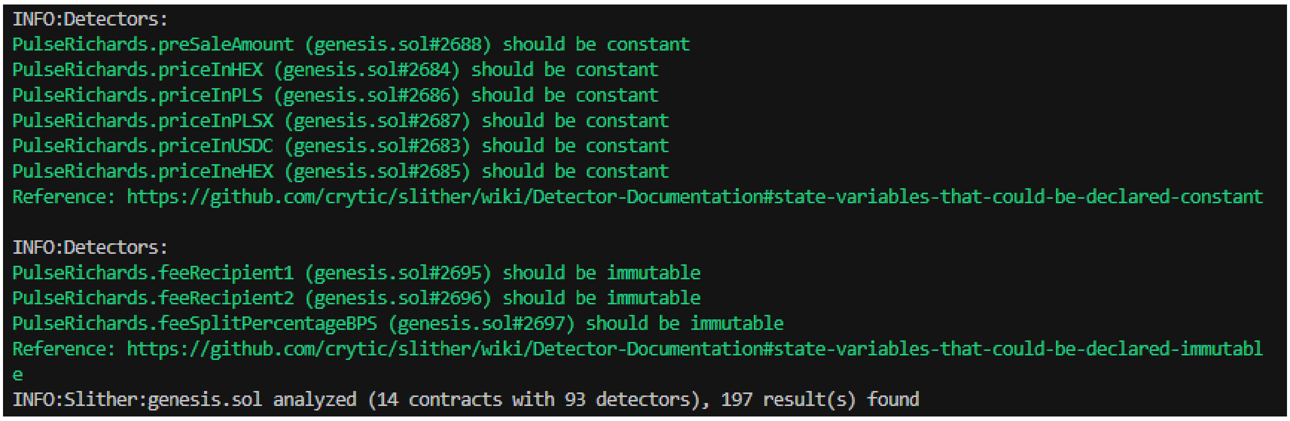
```
LibPRNG.next(LibPRNG.PRNG) (genesis.sol#2187-2203) uses assembly
        - INLINE ASM (genesis.sol#2199-2202)
LibPRNG.uniform(LibPRNG.PRNG,uint256) (genesis.sol#2212-2222) uses assembly
        - INLINE ASM (genesis.sol#2214-2221)
LibPRNG.shuffle(LibPRNG.PRNG,uint256[]) (genesis.sol#2225-2266) uses assembly
        - INLINE ASM (genesis.sol#2227-2265)
LibPRNG.shuffle(LibPRNG.PRNG,bytes) (genesis.sol#2269-2309) uses assembly
        - INLINE ASM (genesis.sol#2271-2308)
LibPRNG.standardNormalWad(LibPRNG.PRNG) (genesis.sol#2312-2334) uses assembly
        - INLINE ASM (genesis.sol#2314-2333)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
LibString.replace(string,string,string) (genesis.sol#1386-1458) has a high cyclomatic complexity (14).
LibString.indexOf(string,string,uint256) (genesis.sol#1463-1516) has a high cyclomatic complexity (13).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity
INFO:Detectors:
Address.functionDelegateCall(address,bytes) (genesis.sol#634-637) is never used and should be removed
Address.functionStaticCall(address,bytes) (genesis.sol#625-628) is never used and should be removed
Address.sendValue(address,uint256) (genesis.sol#571-580) is never used and should be removed
Address.verifyCallResult(bool,bytes) (genesis.sol#665-671) is never used and should be removed
ERC721._burn(uint256) (genesis.sol#2503-2518) is never used and should be removed
ERC721._safeMint(address,uint256) (genesis.sol#2524-2533) is never used and should be removed
```



```
Strings.toHexString(uint256) (genesis.sol#499-503) is never used and should be removed
Strings.toHexString(uint256,uint256) (genesis.sol#508-521) is never used and should be removed
Strings.toStringSigned(int256) (genesis.sol#492-494) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (genesis.sol#2) allows old versions
solc-0.8.22 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (genesis.sol#571-580):
        - (success) = recipient.call{value: amount}() (genesis.sol#576)
Low level call in Address.functionCallWithValue(address,bytes,uint256) (genesis.sol#613-619):
        - (success,returndata) = target.call{value: value}(data) (genesis.sol#617)
Low level call in Address.functionStaticCall(address,bytes) (genesis.sol#625-628):
        - (success,returndata) = target.staticcall(data) (genesis.sol#626)
Low level call in Address.functionDelegateCall(address,bytes) (genesis.sol#634-637):
        - (success,returndata) = target.delegatecall(data) (genesis.sol#635)
Low level call in SafeERC20._callOptionalReturnBool(IERC20,bytes) (genesis.sol#904-911):
        - (success,returndata) = address(token).call(data) (genesis.sol#909)
Low level call in PulseRichards.plsMint(uint256,string) (genesis.sol#2841-2868):
        - (success1) = address(feeRecipient1).call{value: amount1}() (genesis.sol#2852)
        - (success2) = address(feeRecipient2).call{value: amount2}() (genesis.sol#2855)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

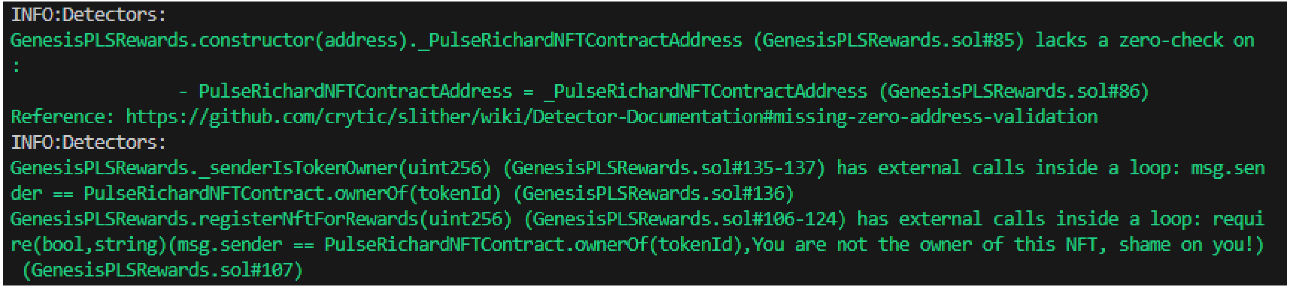


```
INFO:Detectors:
Function IERC20Permit.DOMAIN_SEPARATOR() (genesis.sol#810) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable PulseRichards.constructor(string,string,string,uint256,address,address,uint256)._feeRecipient1 (genesis.sol#271
9) is too similar to PulseRichards.constructor(string,string,string,uint256,address,address,uint256)._feeRecipient2 (gen
esis.sol#2720)
Variable PulseRichards.feeRecipient1 (genesis.sol#2695) is too similar to PulseRichards.feeRecipient2 (genesis.sol#2696)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
LibString.escapeHTML(string) (genesis.sol#1936-1966) uses literals with too many digits:
        - mstore(uint256,uint256)(0x08,0xc0000000a6ab) (genesis.sol#1944)
LibString.escapeHTML(string) (genesis.sol#1936-1966) uses literals with too many digits:
        - ! 1 << c_escapeHTML_asm_0 & 0x500000c400000000 (genesis.sol#1951-1955)
LibPRNG.standardNormalWad(LibPRNG.PRNG) (genesis.sol#2312-2334) uses literals with too many digits:
        - a_standardNormalWad_asm_0 = 0x100000000000000000000000000000051 (genesis.sol#2322)
LibPRNG.standardNormalWad(LibPRNG.PRNG) (genesis.sol#2312-2334) uses literals with too many digits:
        - s_standardNormalWad_asm_0 = 0x1000000000000000100000000000000010000000000000001 (genesis.sol#2324)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

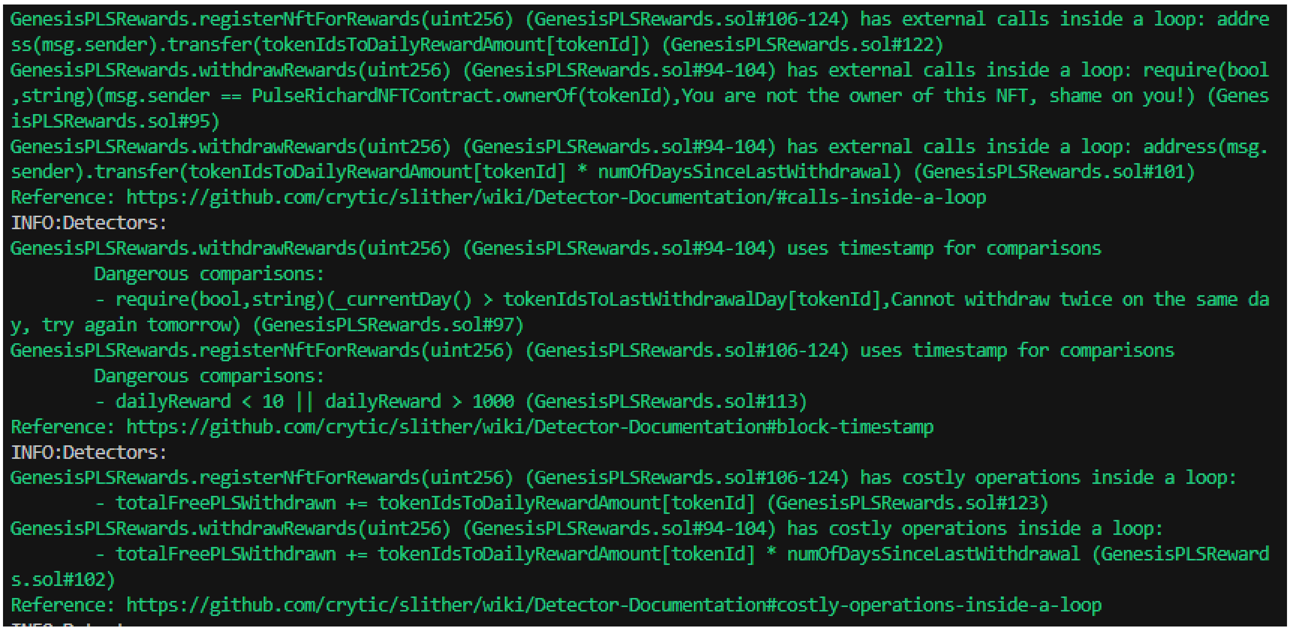


```
INFO:Detectors:
PulseRichards.preSaleAmount (genesis.sol#2688) should be constant
PulseRichards.priceInHEX (genesis.sol#2684) should be constant
PulseRichards.priceInPLS (genesis.sol#2686) should be constant
PulseRichards.priceInPLSX (genesis.sol#2687) should be constant
PulseRichards.priceInUSDC (genesis.sol#2683) should be constant
PulseRichards.priceIneHEX (genesis.sol#2685) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

INFO:Detectors:
PulseRichards.feeRecipient1 (genesis.sol#2695) should be immutable
PulseRichards.feeRecipient2 (genesis.sol#2696) should be immutable
PulseRichards.feeSplitPercentageBPS (genesis.sol#2697) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutabl
e
INFO:Slither:genesis.sol analyzed (14 contracts with 93 detectors), 197 result(s) found
```

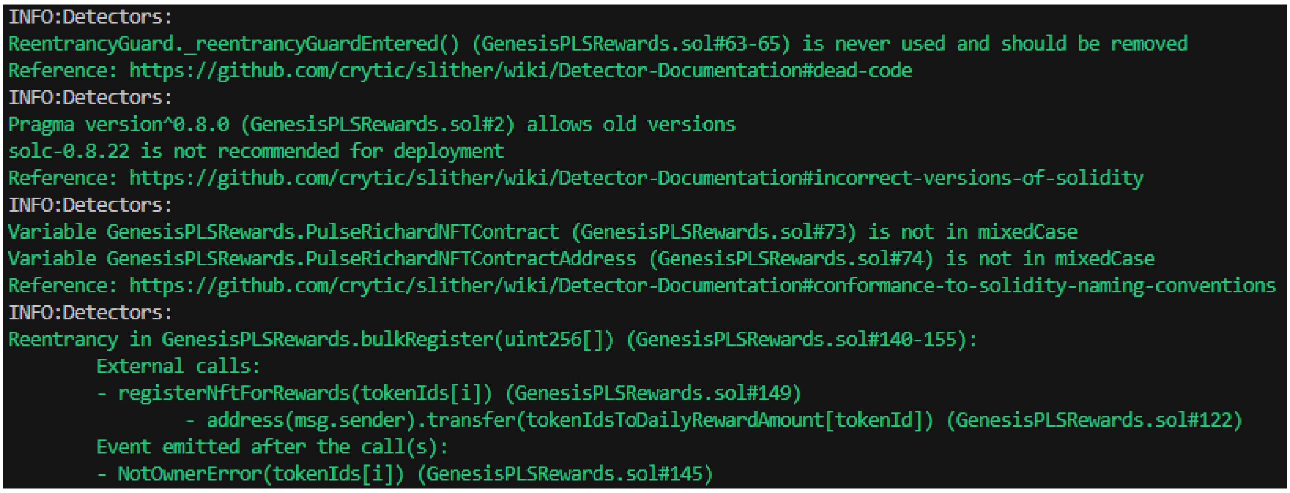
Slither log >> GenesisPLSRewards.sol



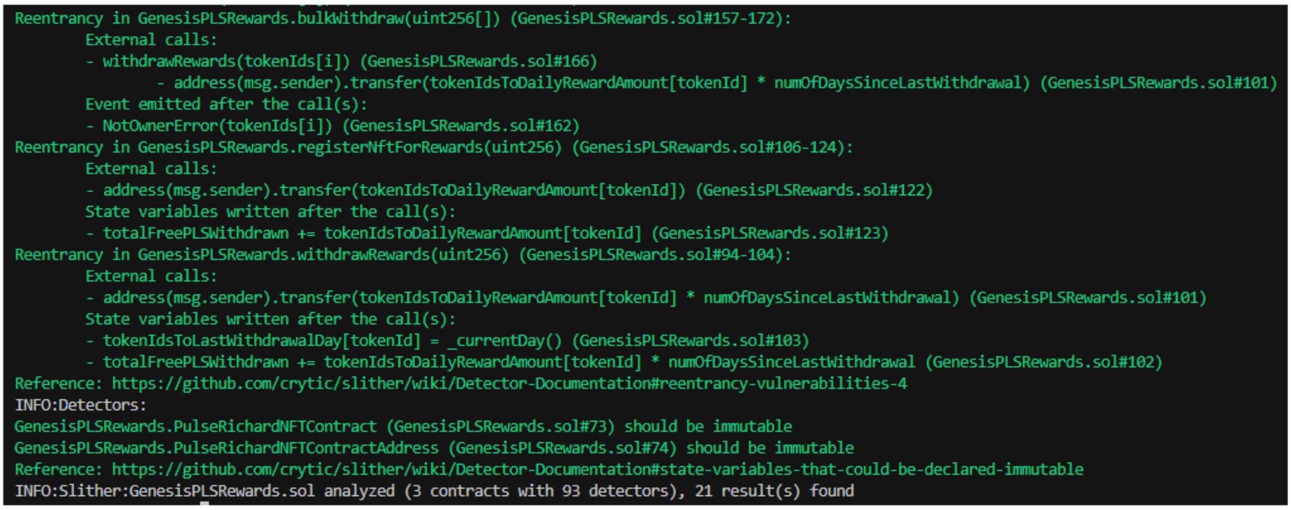
```
INFO:Detectors:
GenesisPLSRewards.constructor(address)._PulseRichardNFTContractAddress (GenesisPLSRewards.sol#85) lacks a zero-check on
:
                - PulseRichardNFTContractAddress = _PulseRichardNFTContractAddress (GenesisPLSRewards.sol#86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
GenesisPLSRewards._senderIsTokenOwner(uint256) (GenesisPLSRewards.sol#135-137) has external calls inside a loop: msg.sen
der == PulseRichardNFTContract.ownerOf(tokenId) (GenesisPLSRewards.sol#136)
GenesisPLSRewards.registerNftForRewards(uint256) (GenesisPLSRewards.sol#106-124) has external calls inside a loop: requi
re(bool,string)(msg.sender == PulseRichardNFTContract.ownerOf(tokenId),You are not the owner of this NFT, shame on you!)
 (GenesisPLSRewards.sol#107)
```



```
GenesisPLSRewards.registerNftForRewards(uint256) (GenesisPLSRewards.sol#106-124) has external calls inside a loop: addre
ss(msg.sender).transfer(tokenIdsToDailyRewardAmount[tokenId]) (GenesisPLSRewards.sol#122)
GenesisPLSRewards.withdrawRewards(uint256) (GenesisPLSRewards.sol#94-104) has external calls inside a loop: require(bool
,string)(msg.sender == PulseRichardNFTContract.ownerOf(tokenId),You are not the owner of this NFT, shame on you!) (Genes
isPLSRewards.sol#95)
GenesisPLSRewards.withdrawRewards(uint256) (GenesisPLSRewards.sol#94-104) has external calls inside a loop: address(msg.
sender).transfer(tokenIdsToDailyRewardAmount[tokenId] * numOfDaysSinceLastWithdrawal) (GenesisPLSRewards.sol#101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
GenesisPLSRewards.withdrawRewards(uint256) (GenesisPLSRewards.sol#94-104) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_currentDay() > tokenIdsToLastWithdrawalDay[tokenId],Cannot withdraw twice on the same da
y, try again tomorrow) (GenesisPLSRewards.sol#97)
GenesisPLSRewards.registerNftForRewards(uint256) (GenesisPLSRewards.sol#106-124) uses timestamp for comparisons
        Dangerous comparisons:
        - dailyReward < 10 || dailyReward > 1000 (GenesisPLSRewards.sol#113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
GenesisPLSRewards.registerNftForRewards(uint256) (GenesisPLSRewards.sol#106-124) has costly operations inside a loop:
        - totalFreePLSWithdrawn += tokenIdsToDailyRewardAmount[tokenId] (GenesisPLSRewards.sol#123)
GenesisPLSRewards.withdrawRewards(uint256) (GenesisPLSRewards.sol#94-104) has costly operations inside a loop:
        - totalFreePLSWithdrawn += tokenIdsToDailyRewardAmount[tokenId] * numOfDaysSinceLastWithdrawal (GenesisPLSReward
s.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```



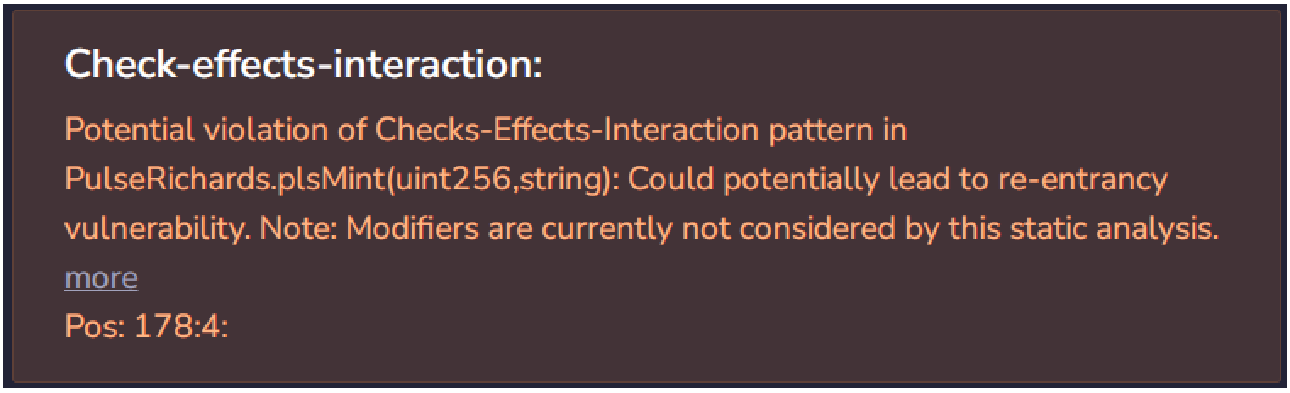
```
INFO:Detectors:
ReentrancyGuard._reentrancyGuardEntered() (GenesisPLSRewards.sol#63-65) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (GenesisPLSRewards.sol#2) allows old versions
solc-0.8.22 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable GenesisPLSRewards.PulseRichardNFTContract (GenesisPLSRewards.sol#73) is not in mixedCase
Variable GenesisPLSRewards.PulseRichardNFTContractAddress (GenesisPLSRewards.sol#74) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in GenesisPLSRewards.bulkRegister(uint256[]) (GenesisPLSRewards.sol#140-155):
        External calls:
        - registerNftForRewards(tokenIds[i]) (GenesisPLSRewards.sol#149)
                - address(msg.sender).transfer(tokenIdsToDailyRewardAmount[tokenId]) (GenesisPLSRewards.sol#122)
        Event emitted after the call(s):
        - NotOwnerError(tokenIds[i]) (GenesisPLSRewards.sol#145)
```



```
Reentrancy in GenesisPLSRewards.bulkWithdraw(uint256[]) (GenesisPLSRewards.sol#157-172):
        External calls:
        - withdrawRewards(tokenIds[i]) (GenesisPLSRewards.sol#166)
                - address(msg.sender).transfer(tokenIdsToDailyRewardAmount[tokenId] * numOfDaysSinceLastWithdrawal) (GenesisPLSRewards.sol#101)
        Event emitted after the call(s):
        - NotOwnerError(tokenIds[i]) (GenesisPLSRewards.sol#162)
Reentrancy in GenesisPLSRewards.registerNftForRewards(uint256) (GenesisPLSRewards.sol#106-124):
        External calls:
        - address(msg.sender).transfer(tokenIdsToDailyRewardAmount[tokenId]) (GenesisPLSRewards.sol#122)
        State variables written after the call(s):
        - totalFreePLSWithdrawn += tokenIdsToDailyRewardAmount[tokenId] (GenesisPLSRewards.sol#123)
Reentrancy in GenesisPLSRewards.withdrawRewards(uint256) (GenesisPLSRewards.sol#94-104):
        External calls:
        - address(msg.sender).transfer(tokenIdsToDailyRewardAmount[tokenId] * numOfDaysSinceLastWithdrawal) (GenesisPLSRewards.sol#101)
        State variables written after the call(s):
        - tokenIdsToLastWithdrawalDay[tokenId] = _currentDay() (GenesisPLSRewards.sol#103)
        - totalFreePLSWithdrawn += tokenIdsToDailyRewardAmount[tokenId] * numOfDaysSinceLastWithdrawal (GenesisPLSRewards.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
GenesisPLSRewards.PulseRichardNFTContract (GenesisPLSRewards.sol#73) should be immutable
GenesisPLSRewards.PulseRichardNFTContractAddress (GenesisPLSRewards.sol#74) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:GenesisPLSRewards.sol analyzed (3 contracts with 93 detectors), 21 result(s) found
```

Solidity Static Analysis

genesis.sol

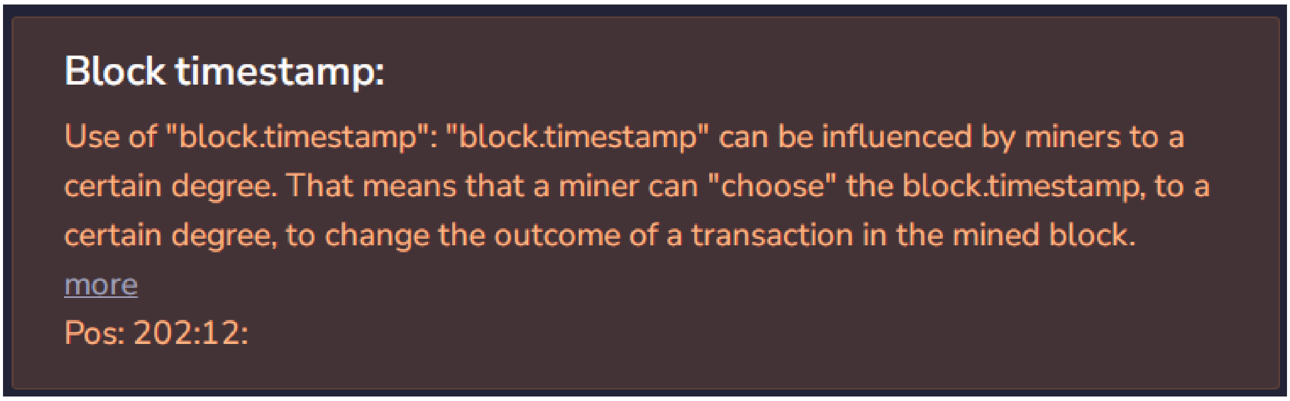


Check-effects-interaction:

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

more

Pos: 178:4:

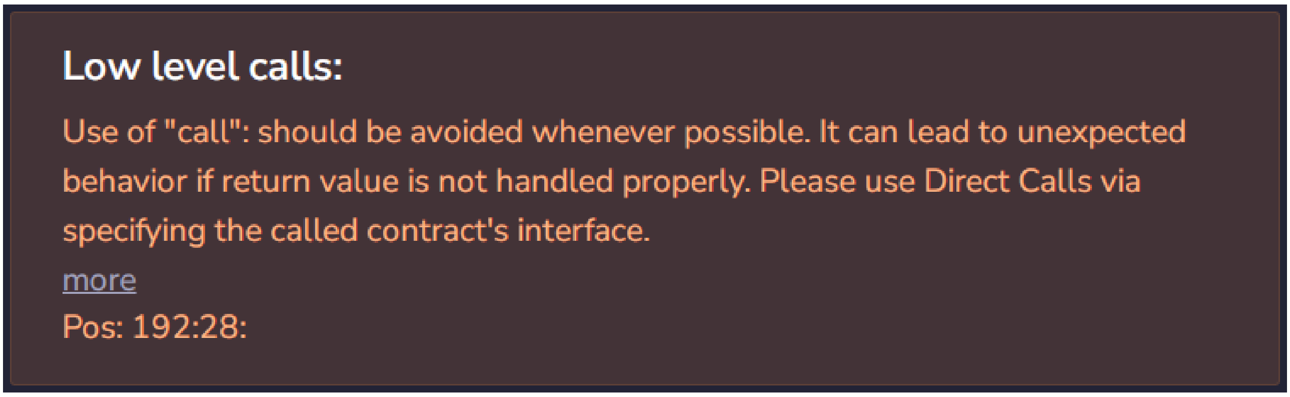


Block timestamp:

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: 202:12:

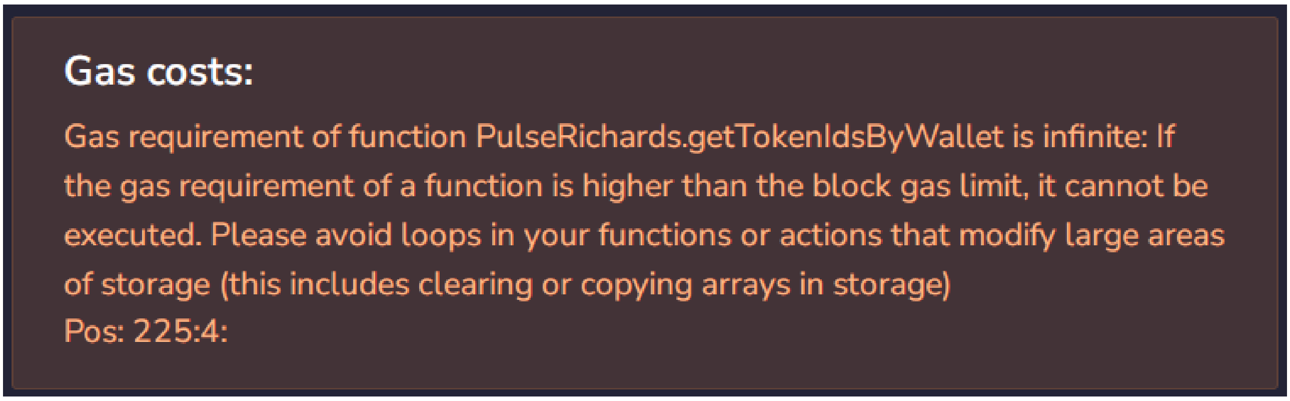


Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

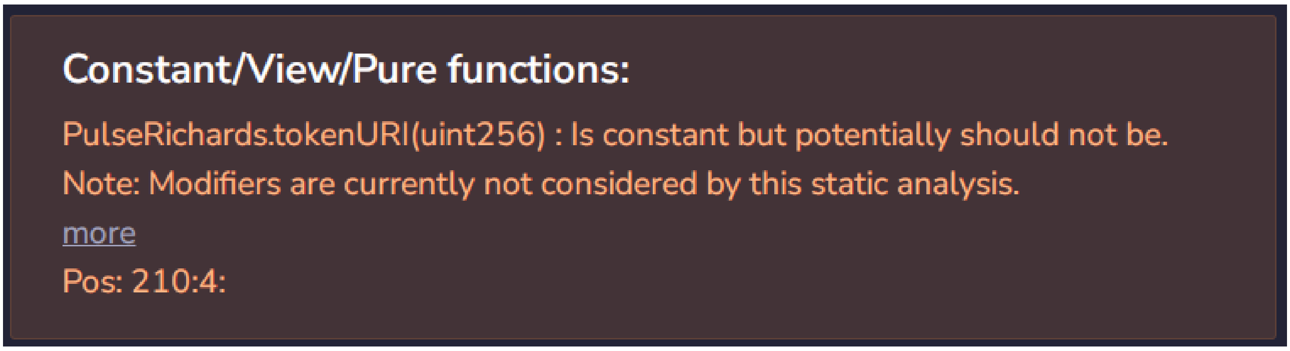
Pos: 192:28:



Gas costs:

Gas requirement of function PulseRichards.getTokenIdsByWallet 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: 225:4:

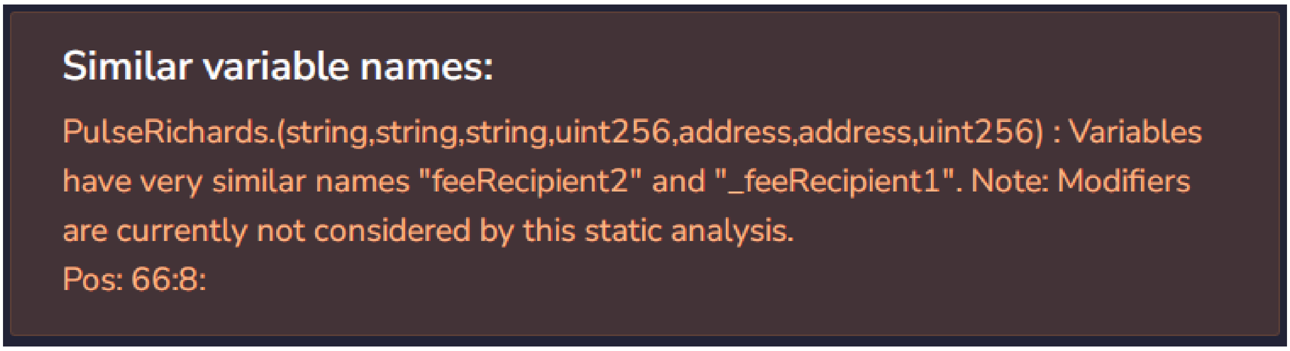


Constant/View/Pure functions:

PulseRichards.tokenURI(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

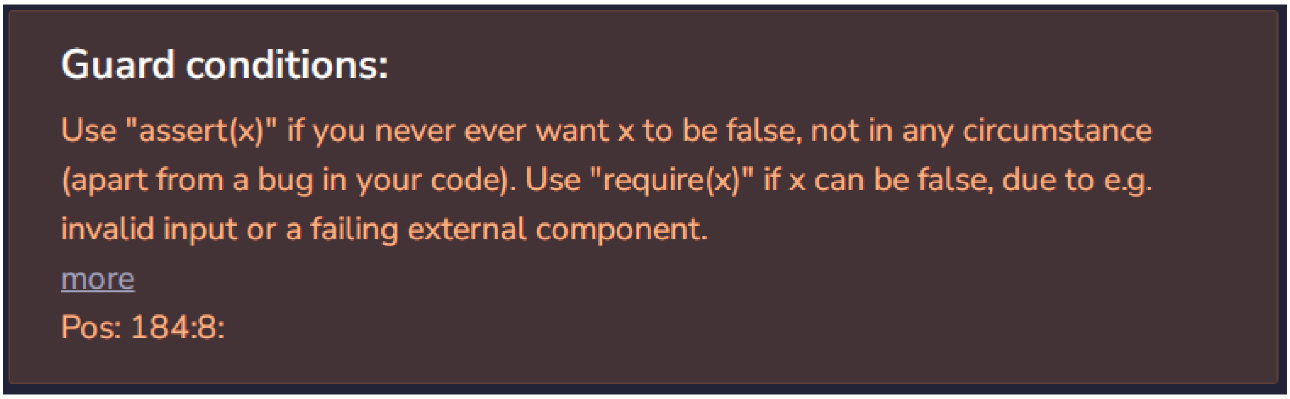
Pos: 210:4:



Similar variable names:

PulseRichards.(string,string,string,uint256,address,address,uint256) : Variables have very similar names "feeRecipient2" and "_feeRecipient1". Note: Modifiers are currently not considered by this static analysis.

Pos: 66:8:

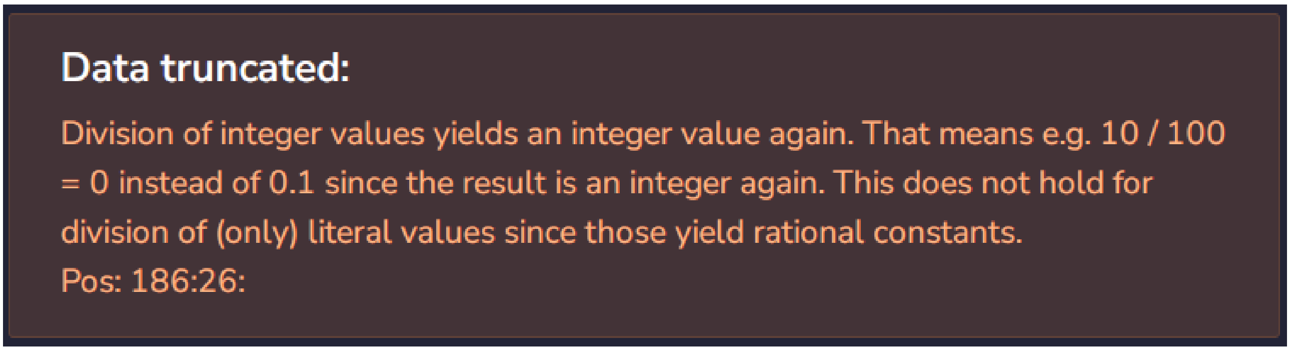


Guard conditions:

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

more

Pos: 184:8:

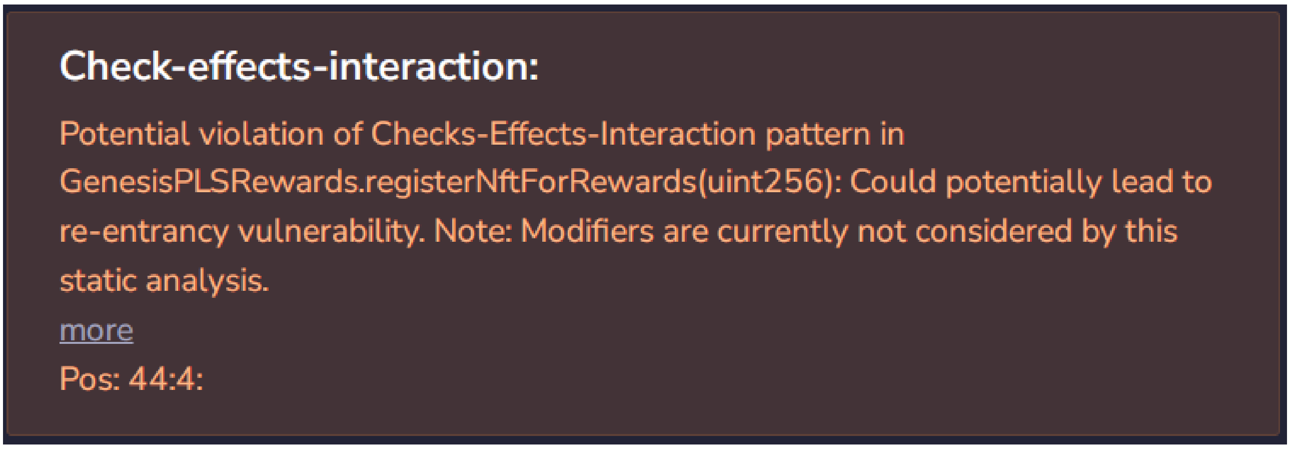


Data truncated:

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

Pos: 186:26:

GenesisPLSRewards.sol

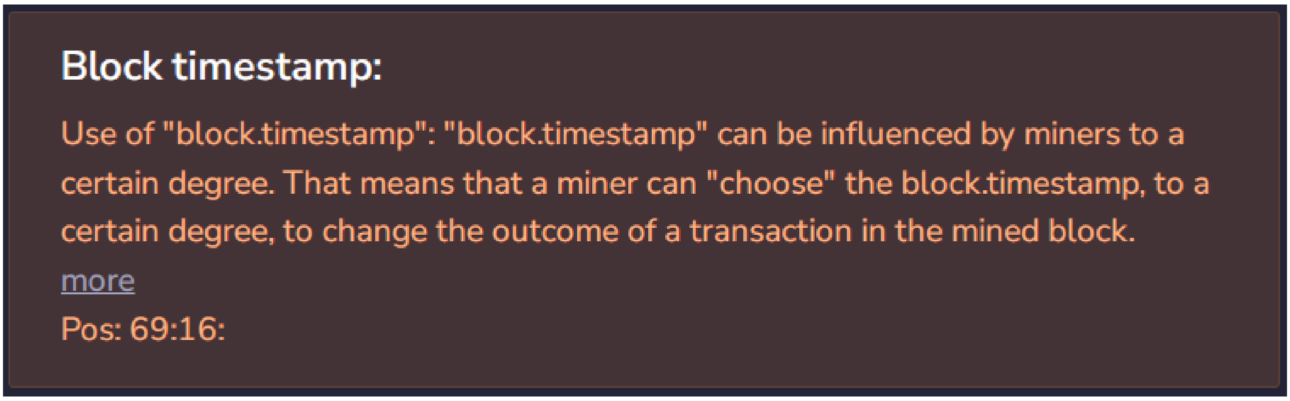


Check-effects-interaction:

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

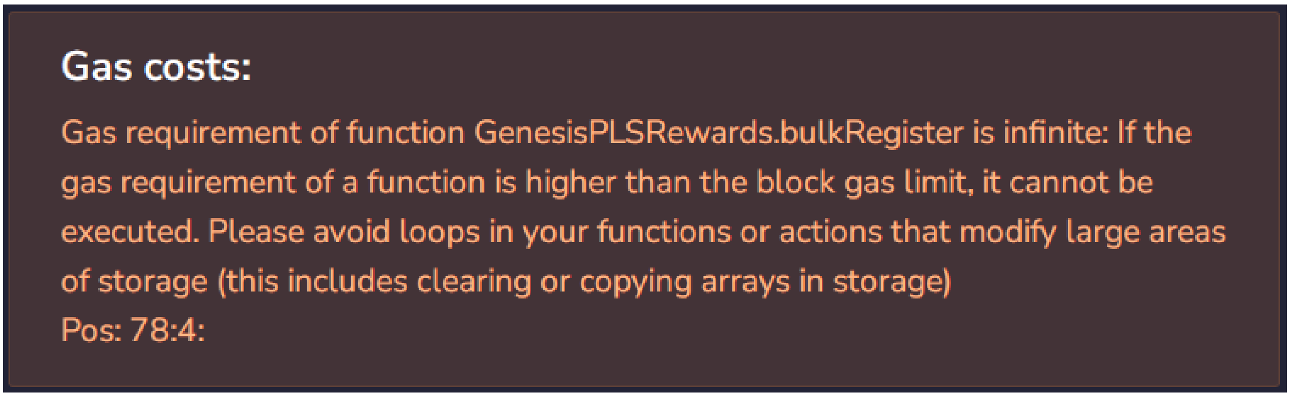
more

Pos: 44:4:



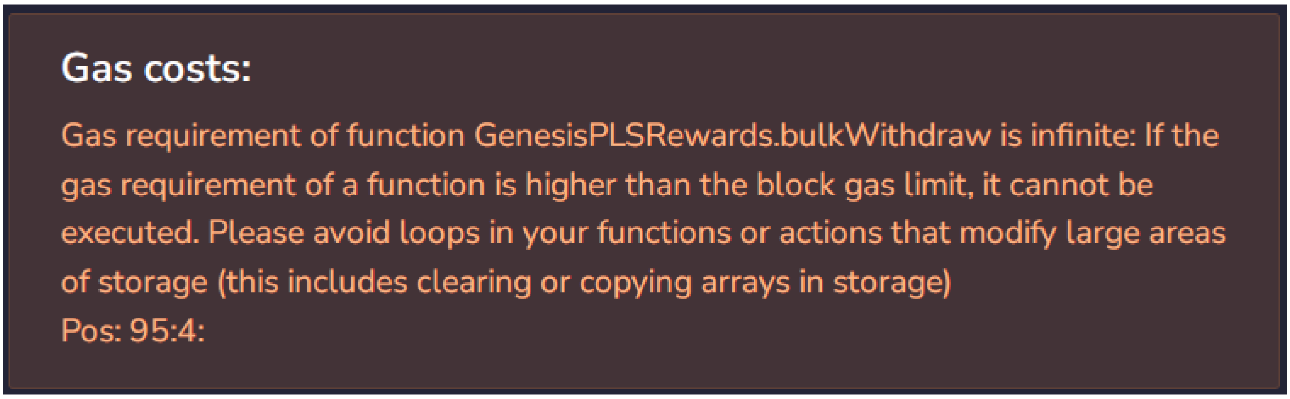
Block timestamp:

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: 69:16:



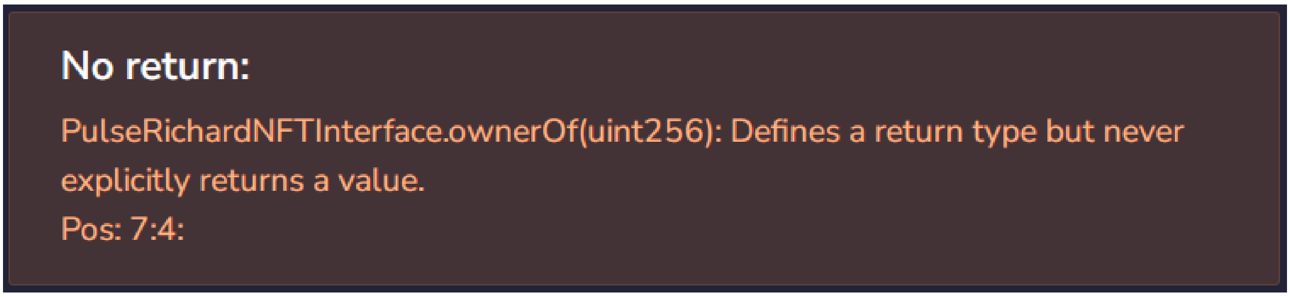
Gas costs:

Gas requirement of function GenesisPLSRewards.bulkRegister 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: 78:4:



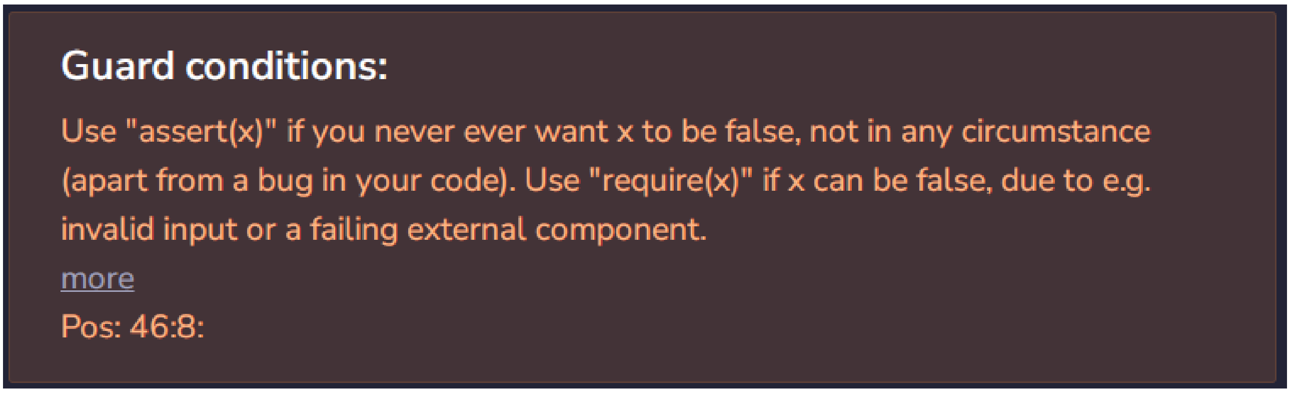
Gas costs:

Gas requirement of function GenesisPLSRewards.bulkWithdraw 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: 95:4:



No return:

PulseRichardNFTInterface.ownerOf(uint256): Defines a return type but never explicitly returns a value.
Pos: 7:4:

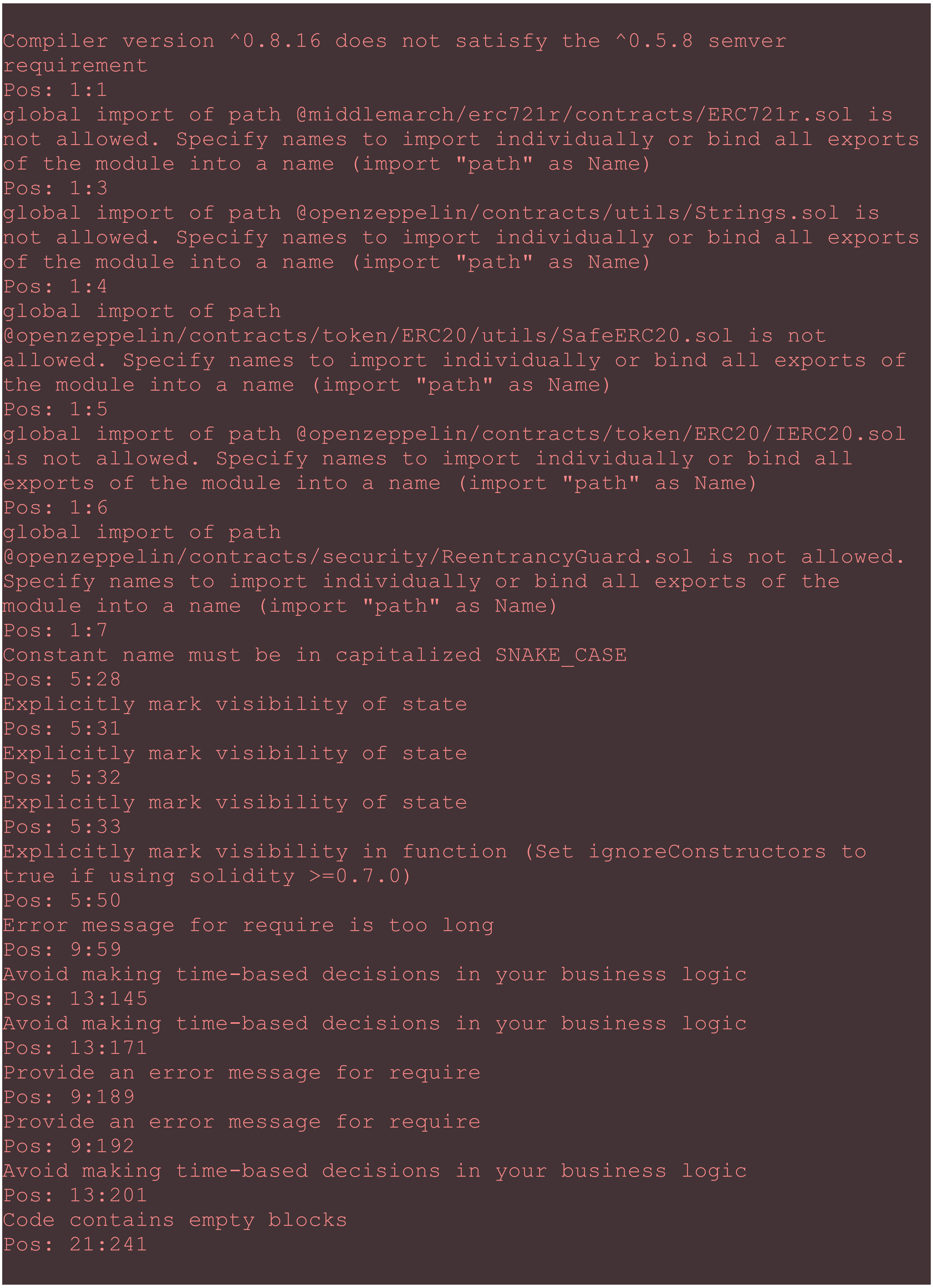


Guard conditions:

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: 46:8:

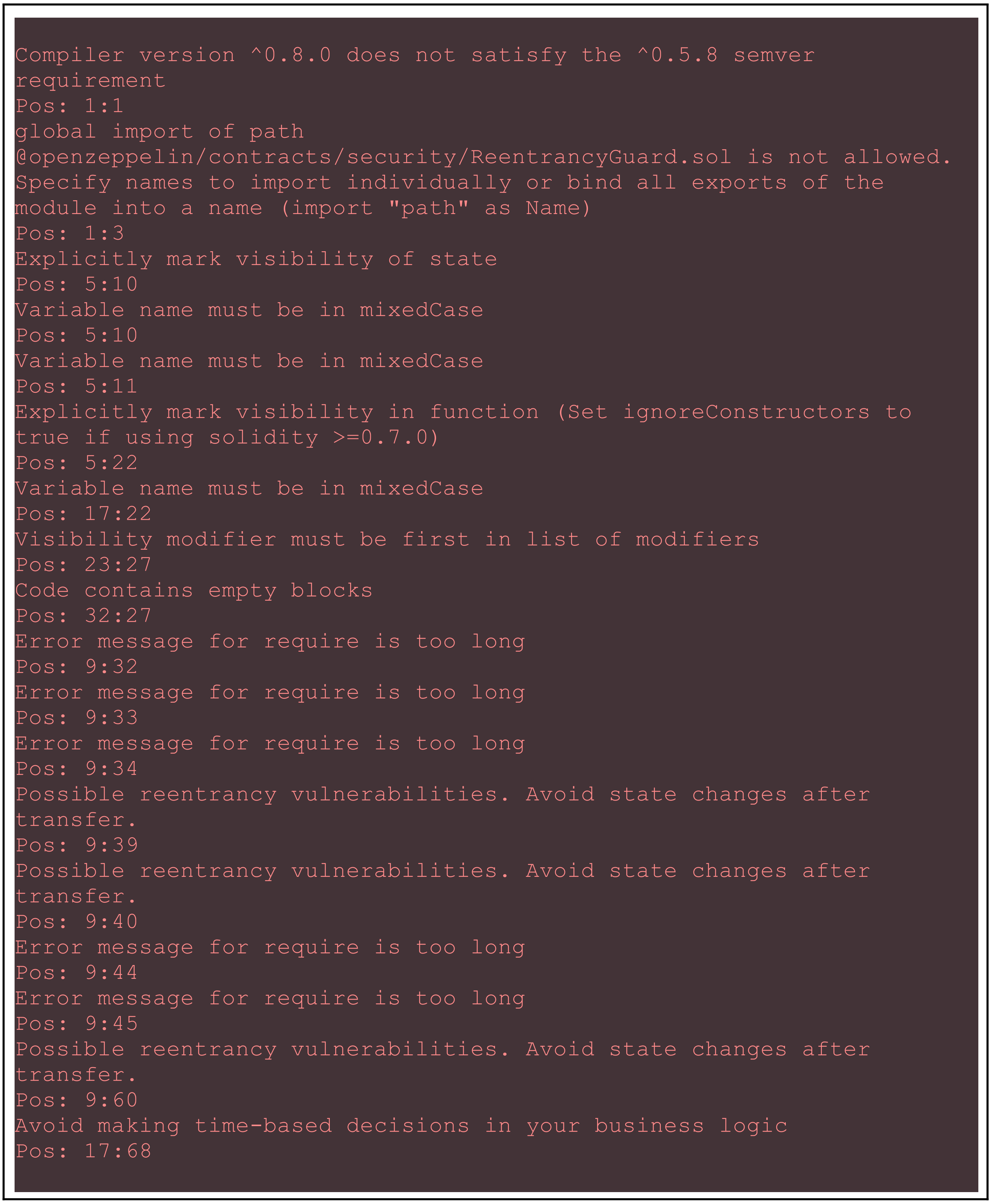
Solhint Linter

PulseRichards.sol



```
Compiler version ^0.8.16 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path @middlemarch/erc721r/contracts/ERC721r.sol is
not allowed. Specify names to import individually or bind all exports
of the module into a name (import "path" as Name)
Pos: 1:3
global import of path @openzeppelin/contracts/utils/Strings.sol is
not allowed. Specify names to import individually or bind all exports
of the module into a name (import "path" as Name)
Pos: 1:4
global import of path
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol is not
allowed. Specify names to import individually or bind all exports of
the module into a name (import "path" as Name)
Pos: 1:5
global import of path @openzeppelin/contracts/token/ERC20/IERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:6
global import of path
@openzeppelin/contracts/security/ReentrancyGuard.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:7
Constant name must be in capitalized SNAKE_CASE
Pos: 5:28
Explicitly mark visibility of state
Pos: 5:31
Explicitly mark visibility of state
Pos: 5:32
Explicitly mark visibility of state
Pos: 5:33
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:50
Error message for require is too long
Pos: 9:59
Avoid making time-based decisions in your business logic
Pos: 13:145
Avoid making time-based decisions in your business logic
Pos: 13:171
Provide an error message for require
Pos: 9:189
Provide an error message for require
Pos: 9:192
Avoid making time-based decisions in your business logic
Pos: 13:201
Code contains empty blocks
Pos: 21:241
```

GenesisPLSRewards.sol



```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path
@openzeppelin/contracts/security/ReentrancyGuard.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:3
Explicitly mark visibility of state
Pos: 5:10
Variable name must be in mixedCase
Pos: 5:10
Variable name must be in mixedCase
Pos: 5:11
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:22
Variable name must be in mixedCase
Pos: 17:22
Visibility modifier must be first in list of modifiers
Pos: 23:27
Code contains empty blocks
Pos: 32:27
Error message for require is too long
Pos: 9:32
Error message for require is too long
Pos: 9:33
Error message for require is too long
Pos: 9:34
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:39
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:40
Error message for require is too long
Pos: 9:44
Error message for require is too long
Pos: 9:45
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:60
Avoid making time-based decisions in your business logic
Pos: 17:68
```

Software analysis result:

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

EtherAuthority

Email: audit@EtherAuthority.io