



**RD
AUDITORS**

Pulse Wars Contract Code Review and Security Analysis Report

Customer: Pulse Wars
Prepared on: 22nd March 2024
Platform: Ethereum
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Document	3
Introduction	5
Project Scope	6
Executive Summary	7
Code Quality	7
Documentation	9
Use of Dependencies	9
AS-IS Overview	10
UML Diagram - Citizens	15
Inheritance Diagram - Citizens	16
Code Flow Diagram - Slither Results Log	17
Audit Findings	23
Conclusion	24
Our Methodology	25
Disclaimers	28

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of Pulse Wars
Platform	Ethereum/ Solidity
File 1	Citizens.sol
MD5 hash	5dad15a78df34f51b293c92768ef1e64
SHA256 hash	88ef307597f51a00116e85dde068fe8c31a456fae1620379b0471d96a7510d44
File 2	PulseWars.sol
MD5 hash	a068fa642a82ef08b6f6efab87895aac
SHA256 hash	109ae7aad0e674b95e52611b8d17380e82ffb62fab967094ca93fb0ac89dbf51
File 3	RandomNumber.sol
MD5 hash	1bc4e639c6390bd07324540cddfe58d0

SHA256 hash	50ffc18a9a4cc09a19b8fcb7a0254c4ebdbf24f253ee3bda47e462ddf4638b4e
Date	22/03/2024

Introduction

RD Auditors (Consultant) were contracted by Pulse Wars (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 16th - 22nd March 2024.

This contract consists of three files.

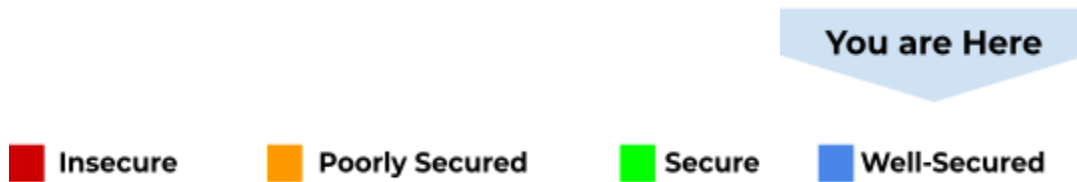
Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level






Executive Summary

According to the assessment, the customer's solidity smart contract is now **Well-Secured**.



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
 Critical	0
 High	0
 Medium	0
 Low	0
 Very Low	0

Code Quality

The libraries within this smart contract are part of a 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 to a specific address and its properties/methods can be reused many times by other contracts.

The Pulse Wars team has provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the Pulse Wars code as a Github link:

<https://github.com/srv-smn/Pulse-Wars/tree/dev>

The hash of that file is mentioned in the table. As mentioned above, it's commented on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

Citizens.sol

File And Function Level Report

Contract: Citizens
Inherit: ICitizens, ERC721, ERC721Enumerable, ReentrancyGuard, Authorizable, ERC2981, Pausable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	changeBaseURI	external	Passed	All Passed	No Issue	Passed
2	setContractURI	external	Passed	All Passed	No Issue	Passed
3	setMaxSupply	public	Passed	All Passed	No Issue	Passed
4	updateThreshold	public	Passed	All Passed	No Issue	Passed
5	updateMintingFee	public	Passed	All Passed	No Issue	Passed
6	updateTreasuryAddress	public	Passed	All Passed	No Issue	Passed
7	updateBaseRoyalty	external	Passed	All Passed	No Issue	Passed
8	removeBaseRoyalty	external	Passed	All Passed	No Issue	Passed

9	updateTokenRoyalty	external	Passed	All Passed	No Issue	Passed
10	removeTokenRoyalty	external	Passed	All Passed	No Issue	Passed
11	updateCoins	public	Passed	All Passed	No Issue	Passed
12	addBadge	public	Passed	All Passed	No Issue	Passed
13	updateCategorySupply	public	Passed	All Passed	No Issue	Passed
14	modifyPartnerStatus	public	Passed	All Passed	No Issue	Passed
15	pauseContract	public	Passed	All Passed	No Issue	Passed
16	unpauseContract	public	Passed	All Passed	No Issue	Passed
17	join	public	Passed	All Passed	No Issue	Passed
18	isWhitelistedPartner	public	Passed	All Passed	No Issue	Passed
19	getCitizenDetails	public	Passed	All Passed	No Issue	Passed
20	getCategoryMaxSupply	external	Passed	All Passed	No Issue	Passed
21	getMaxSupply	external	Passed	All Passed	No Issue	Passed
22	getNFTDetails	public	Passed	All Passed	No Issue	Passed
23	getMintingFees	public	Passed	All Passed	No Issue	Passed
24	getTotalBadgesEarned	public	Passed	All Passed	No Issue	Passed
25	getAllBadges	public	Passed	All Passed	No Issue	Passed
26	getAllCoins	external	Passed	All Passed	No Issue	Passed
27	_beforeTokenTransfer	internal	Passed	All Passed	No Issue	Passed

28	_burn	internal	Passed	All Passed	No Issue	Passed
29	tokenURI	public	Passed	All Passed	No Issue	Passed

PulseWars.sol

Contract: PulseWars
 Inherit: IPulseWars, ERC721, ERC721Enumerable, ReentrancyGuard, Authorizable, ERC2981, Pausable
 Observation: Passed
 Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	changeBaseURI	external	Passed	All Passed	No Issue	Passed
2	setContractURI	external	Passed	All Passed	No Issue	Passed
3	setMaxSupply	public	Passed	All Passed	No Issue	Passed
4	updateThreshold	public	Passed	All Passed	No Issue	Passed
5	updateRandomNumberAddress	public	Passed	All Passed	No Issue	Passed
6	updateMintingFee	public	Passed	All Passed	No Issue	Passed
7	updateTreasuryAddress	public	Passed	All Passed	No Issue	Passed

8	updateRoyalty	external	Passed	All Passed	No Issue	Passed
9	updateCoins	public	Passed	All Passed	No Issue	Passed
10	addBadge	public	Passed	All Passed	No Issue	Passed
11	updateCategorySupply	public	Passed	All Passed	No Issue	Passed
12	pauseContract	public	Passed	All Passed	No Issue	Passed
13	unpauseContract	public	Passed	All Passed	No Issue	Passed
14	getRandomSelection	internal	Passed	All Passed	No Issue	Passed
15	mintNFTs	public	Passed	All Passed	No Issue	Passed
16	getCategoryMaxSupply	external	Passed	All Passed	No Issue	Passed
17	getMaxSupply	external	Passed	All Passed	No Issue	Passed
18	getNFTDetails	public	Passed	All Passed	No Issue	Passed
19	getMintingFees	public	Passed	All Passed	No Issue	Passed
20	getTotalBadgesEarned	public	Passed	All Passed	No Issue	Passed
21	getAllBadges	public	Passed	All Passed	No Issue	Passed
22	getAllCoins	external	Passed	All Passed	No Issue	Passed
23	_baseURI	internal	Passed	All Passed	No Issue	Passed
24	_beforeTokenTransfer	internal	Passed	All Passed	No Issue	Passed
25	_burn	internal	Passed	All Passed	No Issue	Passed
26	tokenURI	public	Passed	All Passed	No Issue	Passed

Code Flow Diagram - Citizens

Please click on the below link to view the image:

https://drive.google.com/file/d/158IA2QeiOIEPtNrzRStvG6CiVIWeESQF/view?usp=drive_link

Code Flow Diagram - Slither Results Log

Citizens.sol

```
INFO:Detectors:
Citizens.modifyPartnerStatus(address,address,bool)._status (Citizens.sol#1361) shadows:
  - ReentrancyGuard._status (Citizens.sol#977) (state variable)
Citizens.join(address,address,uint256)._owner (Citizens.sol#1401) shadows:
  - Ownable._owner (Citizens.sol#1022) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Citizens.setMaxSupply(uint256) (Citizens.sol#1251-1254) should emit an event for:
  - max_total_supply = _maxSupply (Citizens.sol#1253)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (Citizens.sol#154-172) uses assembly
  - INLINE ASM (Citizens.sol#164-167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Address.functionCall(address,bytes) (Citizens.sol#103-105) is never used and should be removed
Address.functionCall(address,bytes,string) (Citizens.sol#107-113) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Citizens.sol#115-121) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Citizens.sol#123-134) is never used and should be removed
Address.functionStaticCall(address,bytes) (Citizens.sol#136-138) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Citizens.sol#140-149) is never used and should be removed
Address.isContract(address) (Citizens.sol#91-94) is never used and should be removed
Address.sendValue(address,uint256) (Citizens.sol#96-101) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (Citizens.sol#154-172) is never used and should be removed
Citizens._beforeTokenTransfer(address,address,uint256,uint256) (Citizens.sol#1513-1520) is never used and should be removed
Context._msgData() (Citizens.sol#387-390) is never used and should be removed
Counter.next(Counter.Counter) (Citizens.sol#59-65) is never used and should be removed
ERC721._baseURI() (Citizens.sol#460-462) is never used and should be removed
ERC721._increaseBalance(address,uint128) (Citizens.sol#584-588) is never used and should be removed
ERC721._safeTransfer(address,address,uint256) (Citizens.sol#707-709) is never used and should be removed
ERC721._safeTransfer(address,address,uint256,bytes) (Citizens.sol#715-718) is never used and should be removed
ERC721._transfer(address,address,uint256) (Citizens.sol#676-686) is never used and should be removed
ERC721Enumerable._addTokenToAllTokensEnumeration(uint256) (Citizens.sol#835-836) is never used and should be removed
ERC721Enumerable._addTokenToOwnerEnumeration(address,uint256) (Citizens.sol#827-829) is never used and should be removed
ERC721Enumerable._removeTokenFromAllTokensEnumeration(uint256) (Citizens.sol#864-874) is never used and should be removed
```

```
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (Citizens.sol#218-225) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (Citizens.sol#179-185) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (Citizens.sol#187-194) is never used and should be removed
Strings.toHexString(address) (Citizens.sol#277-279) is never used and should be removed
Strings.toHexString(uint256) (Citizens.sol#265-267) is never used and should be removed
Strings.toHexString(uint256,uint256) (Citizens.sol#269-275) is never used and should be removed
Strings.toString(uint256) (Citizens.sol#261-263) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version0.8.20 (Citizens.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.20 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) (Citizens.sol#96-101):
- (success) = recipient.call{value: amount}{} (Citizens.sol#99)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Citizens.sol#123-134):
- (success,returndata) = target.call{value: value}(data) (Citizens.sol#132)
Low level call in Address.functionStaticCall(address,bytes,string) (Citizens.sol#140-149):
- (success,returndata) = target.staticcall(data) (Citizens.sol#147)
Low level call in Citizens.join(address,address,uint256) (Citizens.sol#1390-1411):
- (sent) = treasury.call{value: amountToPay}{} (Citizens.sol#1409)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Event ICitizens.modifyPartners(address,bool) (Citizens.sol#18) is not in CapWords
Parameter ICitizens.join(address,address,uint256).ID (Citizens.sol#21) is not in mixedCase
Parameter Authorizable.addAuthorized(address)._toAdd (Citizens.sol#1153) is not in mixedCase
Parameter Authorizable.removeAuthorized(address)._toRemove (Citizens.sol#1158) is not in mixedCase
Parameter Citizens.changeBaseURI(string)._baseURL (Citizens.sol#1234) is not in mixedCase
Parameter Citizens.setContractURI(string)._contractURI (Citizens.sol#1243) is not in mixedCase
Parameter Citizens.setMaxSupply(uint256)._maxSupply (Citizens.sol#1251) is not in mixedCase
Parameter Citizens.updateMintingFee(uint256,uint256)._cMintingFee (Citizens.sol#1268) is not in mixedCase
Parameter Citizens.updateMintingFee(uint256,uint256)._pMintingFee (Citizens.sol#1268) is not in mixedCase
Parameter Citizens.updateTreasuryAddress(address)._treasury (Citizens.sol#1281) is not in mixedCase
Parameter Citizens.updateBaseRoyalty(address,uint96)._royaltyReceiver (Citizens.sol#1295) is not in mixedCase
```

```
Parameter Citizens.join(address,address,uint256)._to (Citizens.sol#1391) is not in mixedCase
Parameter Citizens.join(address,address,uint256)._partner (Citizens.sol#1392) is not in mixedCase
Parameter Citizens.join(address,address,uint256).ID (Citizens.sol#1393) is not in mixedCase
Parameter Citizens.isWhitelistedPartner(address)._partner (Citizens.sol#1420) is not in mixedCase
Parameter Citizens.getTotalBadgesEarned(uint256)._tokenId (Citizens.sol#1462) is not in mixedCase
Parameter Citizens.getAllBadges(uint256,uint256,uint256)._tokenId (Citizens.sol#1478) is not in mixedCase
Parameter Citizens.getAllBadges(uint256,uint256,uint256)._startIndex (Citizens.sol#1479) is not in mixedCase
Parameter Citizens.getAllBadges(uint256,uint256,uint256)._endIndex (Citizens.sol#1480) is not in mixedCase
Parameter Citizens.getAllCoins(uint256)._tokenId (Citizens.sol#1499) is not in mixedCase
Variable Citizens.max_total_supply (Citizens.sol#1199) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Citizens.sol#388)" inContext (Citizens.sol#382-391)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
INFO:Detectors:
Variable Citizens.updateMintingFee(uint256,uint256)._cMintingFee (Citizens.sol#1268) is too similar to Citizens.updateMintingFee(uint256,uint256)._pMintingFee (Citizens.sol#1268)
Variable Citizens.setContractURI(string)._contractURI (Citizens.sol#1243) is too similar to Citizens.constructor(string,string,address,uint96)._contracturl (Citizens.sol#1203)
Variable Citizens.cMintingFee (Citizens.sol#1187) is too similar to Citizens.pMintingFee (Citizens.sol#1188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
Citizens (Citizens.sol#1169-1540) does not implement functions:
  - ICitizens.getCitizenDetails(uint256) (Citizens.sol#34-35)
  - ICitizens.modifyPartnerStatus(address,bool) (Citizens.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Detectors:
Citizens.defaultRoyalty (Citizens.sol#1191) should be immutable
Citizens.threshold (Citizens.sol#1189) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:Citizens.sol analyzed (26 contracts with 93 detectors), 75 result(s) found
```

Solidity Static Analysis

Citizens.sol

Check-effects-interaction:

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

[more](#)

Pos: 258:4:

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: 279:24:

Constant/View/Pure functions:

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

[more](#)

Pos: 372:4:

Similar variable names:

Citizens.join(address,address,uint256) : Variables have very similar names "_owner" and "_owners". Note: Modifiers are currently not considered by this static analysis.
Pos: 270:16:

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: 377:8:

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: 379:8:

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Discussion

Conclusion

We were given a contract file and have used all possible tests based on the given object. so it is ready for mainnet deployment. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is “**Well-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 analyse 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 scrutinised 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

RD Auditors Disclaimer

The smart contracts given for audit have been analysed 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the 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 recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. 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.



Email: info@rdauditors.com

Website: www.rdauditors.com

