SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

PerpInu

Audit

Security Assessment

24.November,2022

For

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	23.November,2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Ethereum (ETH)

Website
https://perpinu.finance/

Twitter
https://twitter.com/PerpInu

Telegram
https://t.me/perpinuportal

Medium
https://medium.com/@PerpInu

Discord
https://discord.com/invite/perpinu

Description

Perp Inu combines the fun of a meme-token, the utility of a toolbox and the rewards of decentralization.
Our main goal is to contribute to the Arbitrum ecosystem with useful tools for all its users. From project creators to the most degenerate members of the community.

Project Engagement

During the 23rd of November 2022, **PerpInu** team engaged Solidproof.io to audit the smart contracts that they created. The engagement was technical in nature and focused on identifying the security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Links

v1.0

https://goerli.etherscan.io/address/0x82F3332f852fD46c9faC4aF8c0522d72520C90eB#code

Implementation:

https://goerli.etherscan.io/address/0x80ec326fc529675908db7054c292822d083e2643#code

Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
    ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.

2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analyzing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (Direct imports)

Imported packages:

Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	1
@openzeppelin/contracts/access/Ownable.sol	2
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol	1
@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol	1

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

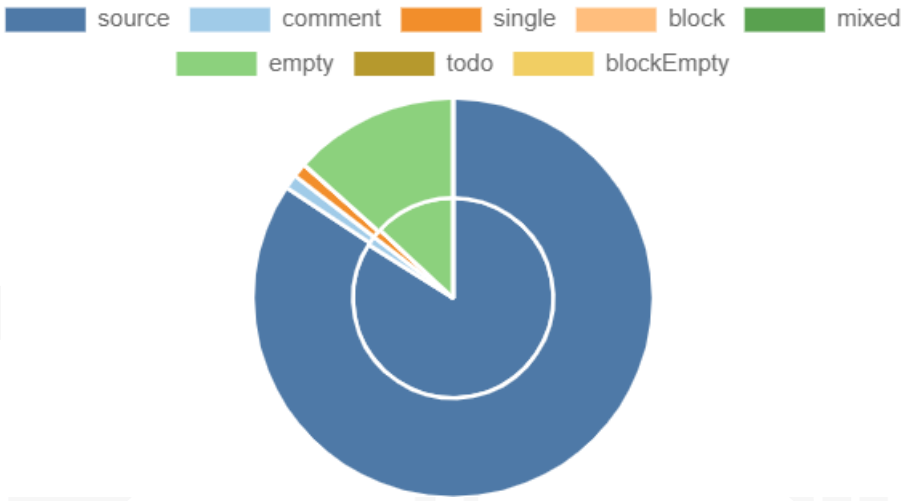
v1.0

File Name	SHA-1 Hash
contracts/LiquidityLocker.sol	e2aa278918a273937ee1abd85ae6369a5a3590ad
contracts/Helpers/IWETH.sol	265ec3a2540ab4abb72af7cebbea21edc4ee0691
contracts/BuyerLocker.sol	61665657f99fd2dd8a90cfe174be7f8a21b2ebd6
contracts/INonfungiblePositionManager.sol	5ba3ef326b523dd6067d93af7cac96576f031bdf
contracts/BaseLocker.sol	1bafd8caca5d45f5bb5cf144cd82e97fde05b101

Metrics

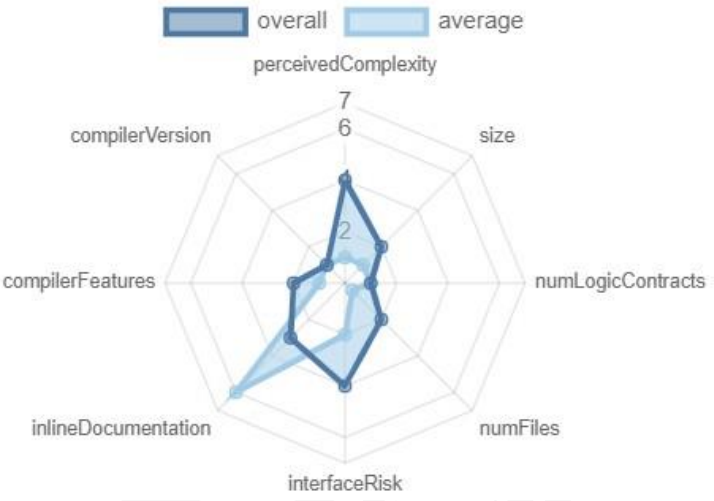
Source Lines

v1.0



Risk Level

v1.0



Capabilities

v1.0

Components

Contracts	Libraries	Interfaces	Abstract
3	0	2	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
34	7

External	Internal	Private	Pure	View
30	36	2	0	11

StateVariables

Total	Public
10	9

Capabilities

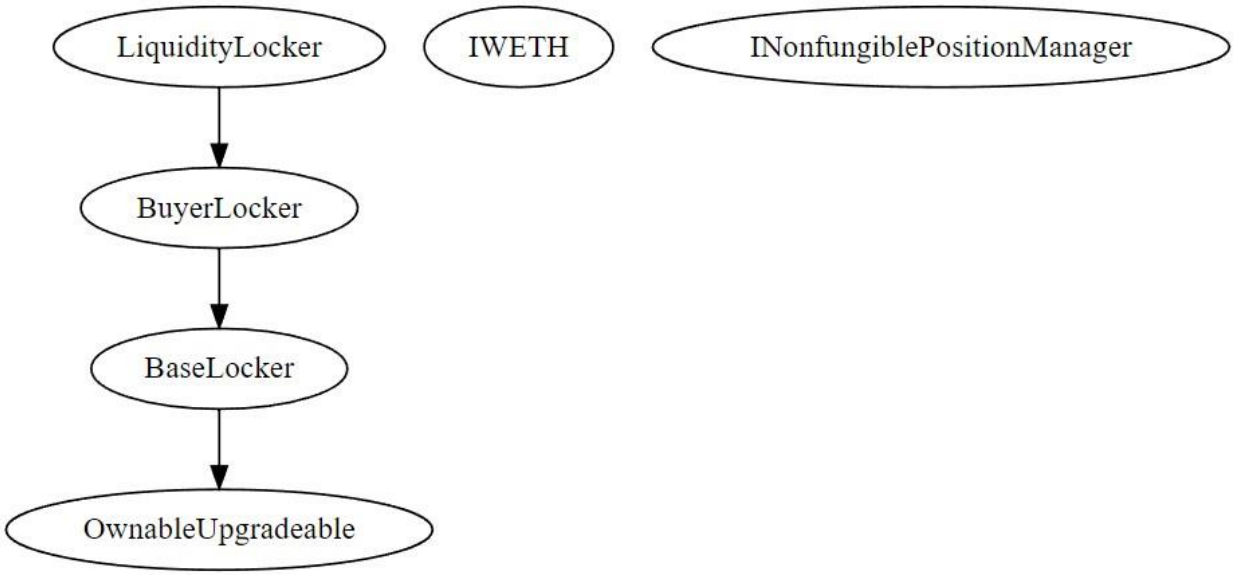
Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`^0.8.4`		`yes`		

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
`yes`					

TryCatch	Σ Unchecked

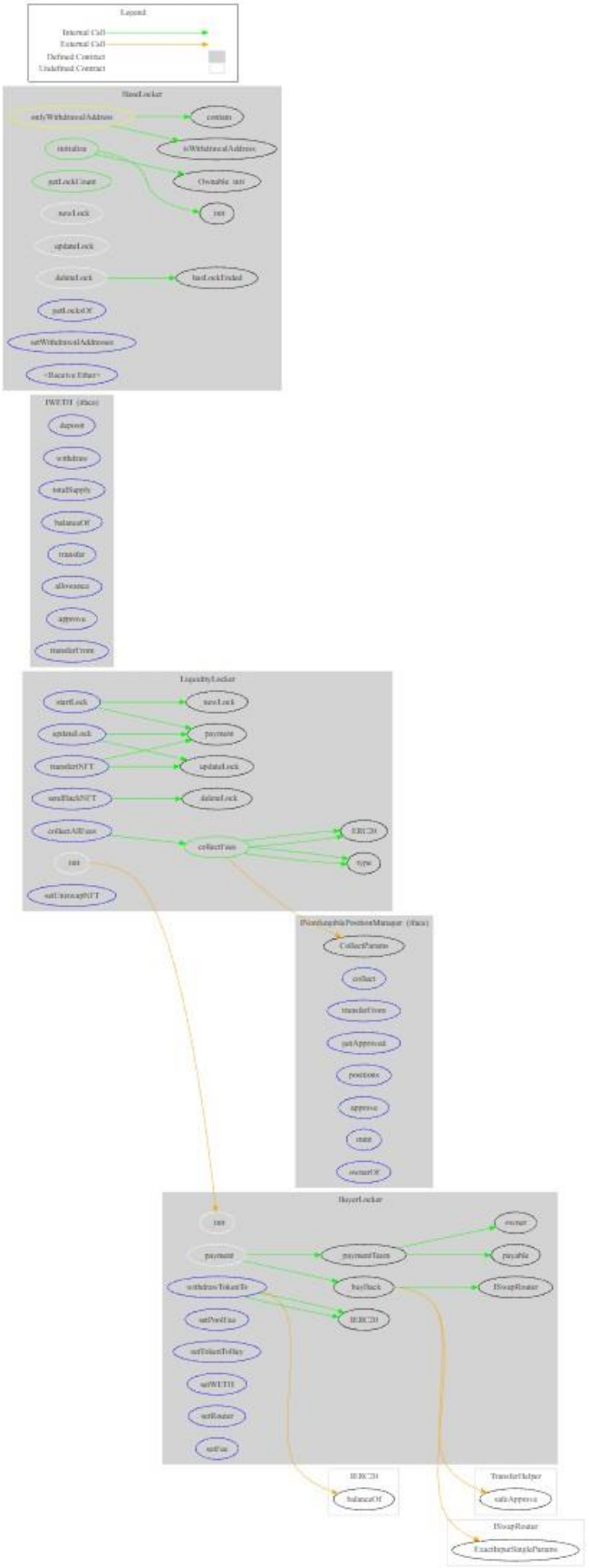
Inheritance Graph

v1.0



Call Graph

v1.0



Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (GitHub, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Is contract an upgradeable
2. Correct implementation of Token standard
3. Deployer cannot mint any new tokens
4. Deployer cannot burn or lock user funds
5. Deployer cannot pause the contract
6. Deployer can set fees
7. Deployer can blacklist/antisnipe address
8. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	YES

Comments:

- The owner can change the Implementation contract address anytime

Correct implementation of Token standard

ERC20				
Function	**Description**	**Exist**	**Tested**	**Verified**
totalSupply	Provides information about the total token supply			
balanceOf	Provides account balance of the owner's account			
transfer	Executes transfers of a specified number of tokens to a specified address			
transferFrom	Executes transfers of a specified number of tokens from a specified address			
approve	Allow a spender to withdraw a set number of tokens from a specified account			
allowance	Returns a set number of tokens from a spender to the owner			

Write functions of contracts

v1.0

1. collectAllFees (0x686f2c90)
2. collectFees (0xb17acdcd)
3. initialize (0x8129fc1c)
4. renounceOwnership (0x715018a6)
5. sendBackNFT (0x17431506)
6. setFee (0x69fe0e2d)
7. setPoolFee (0x73dd250c)
8. setRouter (0xc0d78655)
9. setTokenToBuy (0xc307736b)
10. setUniswapNFT (0x44853a2b)
11. setWETH (0x5b769f3c)
12. setWithdrawalAddresses (0xffe6a117)
13. startLock (0x45bf4dd0)
14. transferOwnership (0xf2fde38b)
15. transfertNFT (0xac6a9dd2)
16. updateLock (0x5a2aaaa4)
17. withdrawTokenTo (0x338b5f0d)

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint			
Max / Total Supply	N/A		

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer can lock			
Deployer cannot burn			

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer can pause			

Deployer can set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 25%			
Deployer cannot set fees to nearly 100%			

Comments:

- The owner can set fees to any arbitrary amount in the BuyerLocker contract. Moreover, the fees can be set to 100% or more

Deployer cannot blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer can blacklist/antisnipe addresses			

Overall checkup (Smart Contract Security)

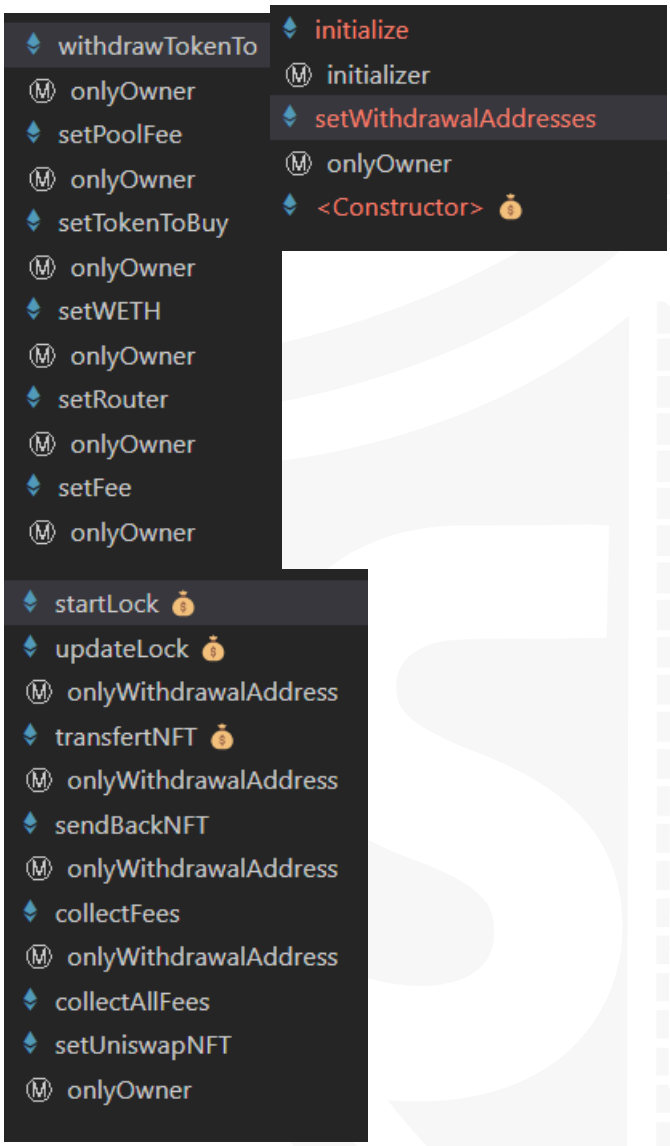
Tested	Verified

Legend

Attribute	Symbol
Verified / Checked	
Partly Verified	
Unverified / Not checked	
Not available	

Modifiers and public functions

v1.0



Ownership Privileges:

The contract has two authorities that can make critical parameter changes in the contract. First is the owner itself and the other are the accounts that are in the “WithdrawalAddress” list.

- Set withdrawal, Token to buy, WETH, and Router address
- Withdraw tokens to another contract.
- Set Uniswap NFT address

The withdrawal addresses added by the owner have the privileges mentioned below:

- Update the lock, Transfer NFT
- Send Back NFT, and Collect fees

Source Units in Scope

v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/LiquidityLocker.sol	1	————	114	99	87	1	73
contracts/Helpers/IWETH.sol	————	1	30	5	3	1	20
contracts/BuyerLocker.sol	1	————	89	89	76	1	51
contracts/INonfungiblePositionManager.sol	————	1	69	26	22	1	21
contracts/BaseLocker.sol	1	————	127	113	95	1	50
Totals	**3**	**2**	**429**	**332**	**283**	**5**	**215**

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments

Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

AUDIT PASSED

Critical issues

No Critical issues

High issues

No High issues

Medium issues

No Medium issues

Low issues

Issue	File	Type	Line	Description
#1	BuyerLocker	Missing zero address validation	58,74-82	We recommend to check that the passed address is not zero
#2	BuyerLocker/ LiquidityLocker	Missing Events	All	Emit events for critical parameter changes. There are no events in the contract
#3	All	Floating Pragma	-	The current pragma Solidity directive is “^0.8.4”. Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma

				helps to ensure that contracts do not accidentally get deployed using other versions.
#4	BseLocker	Missing Events	122	Emit events for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

24. November, 2022:

- There is still an owner (Owner still has not renounced ownership).
- There is an Interface used in the contract and we were not provided with the actual code of that interface so we cannot comment anything regarding the security of "INonfungiblePositionManager.sol" file.
- Read the whole report and modifiers section for more information.

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED

131			
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SWC-	Write to Arbitrary	CWE-123: Write-what-where Condition	PASSED

124	Storage Location		
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED

SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED

SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED

SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED

SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED