



**SOLID**Proof  
*Bring trust into your projects*

**Blockchain Security | Smart Contract Audits | KYC**

MADE IN GERMANY

# CryptoBots Audit

**Security Assessment**  
13.August,2022

**For**



[SolidProof.io](https://solidproof.io)



[@solidproof\\_io](https://t.me/solidproof_io)

Disclaimer	2
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Inheritance Graph	12
CallGraph	13
Scope of Work/Verify Claims	14
Modifiers and public functions	21
Source Units in Scope	22
Critical issues	23
High issues	23
Medium issues	23
Low issues	23
Informational issues	24
Audit Comments	24
SWC Attacks	25

## 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	11.August,2022	<ul style="list-style-type: none"><li>• Layout project</li><li>• Automated- /Manual-Security Testing</li><li>• Summary</li></ul>

## Network

Ethereum (ETH)

## Website

<https://cryptobots.me/>

## Twitter

[https://twitter.com/cryptobots\\_game](https://twitter.com/cryptobots_game)

## Telegram

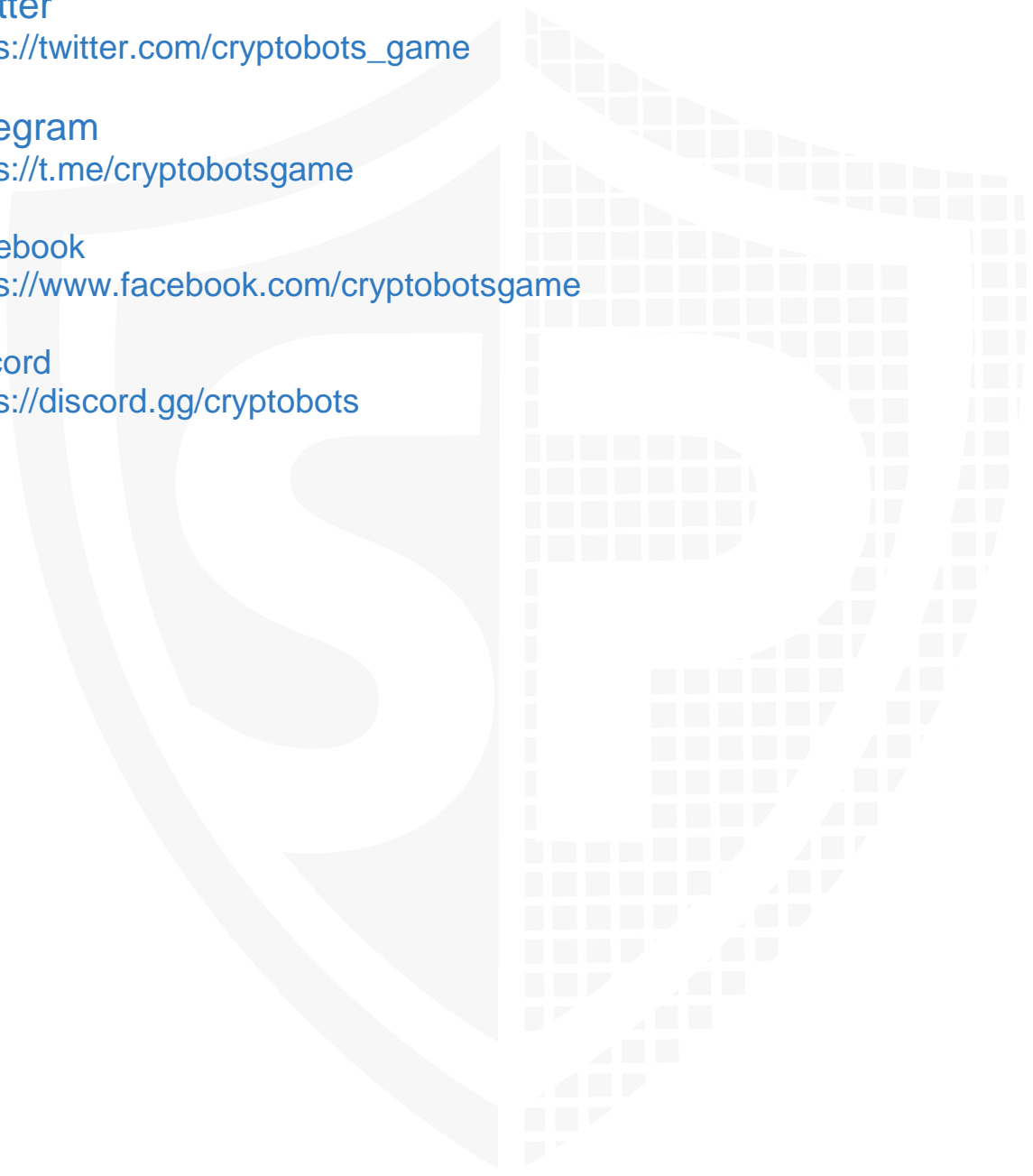
<https://t.me/cryptobotsgame>

## Facebook

<https://www.facebook.com/cryptobotsgame>

## Discord

<https://discord.gg/cryptobots>



## Description

We at Playneta studios created Cryptobots in 2017 with a mission to bring real ownership to games.

Utilizing our exceptional expertise, we wasted no time developing the world's most engaging gameplay yielding unmatched real-world rewards.

With all the latest advancements, we're bringing forward Cryptobots Metaverse: a gaming experience of galactic proportions!

The game evolves and expands on the core concept, shifting from the core gameplay that people love to a space strategy sim with multi-level alliances, wars, power struggles, resource markets, and most importantly real ownership and economy.

Using the tried-and-true formulas from our popular games, our vast expertise in game dev, and operating a crypto game along with the breakthroughs in blockchain technologies we act on the verge of a whole new paradigm of gaming.

## Project Engagement

During the 11<sup>th</sup> of August 2022, **CryptoBots** 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://etherscan.io/address/0x98993f2FDE60c6df2Ff4Db5594201c3b7dbfe23B#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)
<b>Critical</b>	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.
<b>High</b>	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.
<b>Medium</b>	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.
<b>Low</b>	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.
<b>Informational</b>	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.

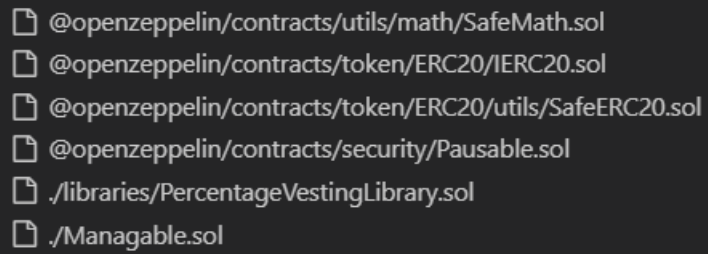
## **Methodology**

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:



- 📄 @openzeppelin/contracts/utils/math/SafeMath.sol
- 📄 @openzeppelin/contracts/token/ERC20/IERC20.sol
- 📄 @openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
- 📄 @openzeppelin/contracts/security/Pausable.sol
- 📄 ./libraries/PercentageVestingLibrary.sol
- 📄 ./Managable.sol



## Tested Contract Files

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

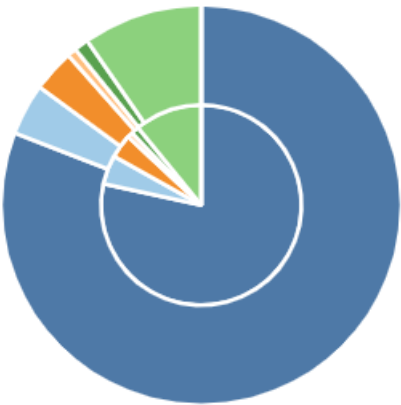
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

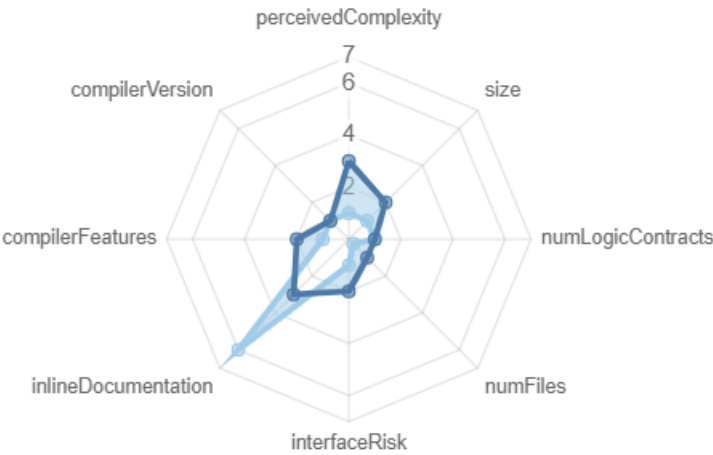
File Name	SHA-1 Hash
contracts/Vesting.sol	9dcb00f770d08777d999099e6e8977b12f26d344
contracts/Managable.sol	4eb6137ad549145b241b3d90fb666de4f19c1dfe
contracts/libraries/BP.sol	95bad10dab68a169c2fafd014032b4b5c659f745
contracts/libraries/PercentageVestingLibrary.sol	4be843bf2661f8e0638b7cdb3bfa6df3936209c3

# Metrics

## Source Lines v1.0



## Risk Level v1.0



## Capabilities

### Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	2	0	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.*

Version	Public	Payable
1.0	16	0

Version	External	Internal	Private	Pure	View
1.0	12	19	1	0	9

### State Variables

Version	Total	Public
1.0	9	8

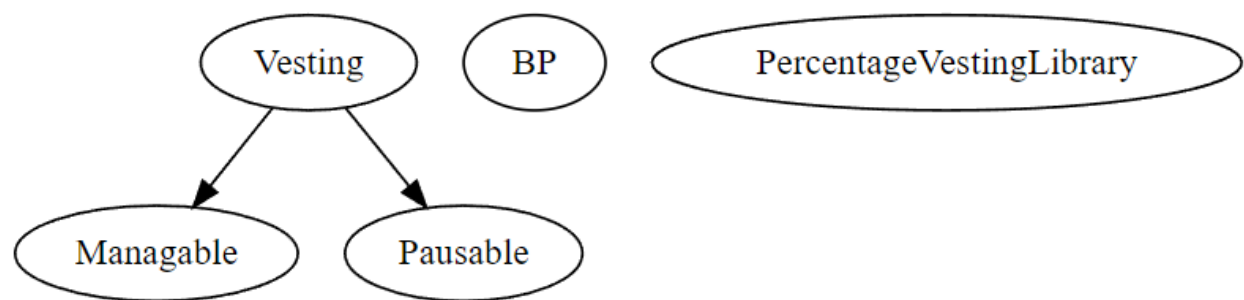
### Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	<code>^0.8.0</code>				

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/Create/Create2
1.0						

# Inheritance Graph

v1.0



# Call Graph

## v1.0

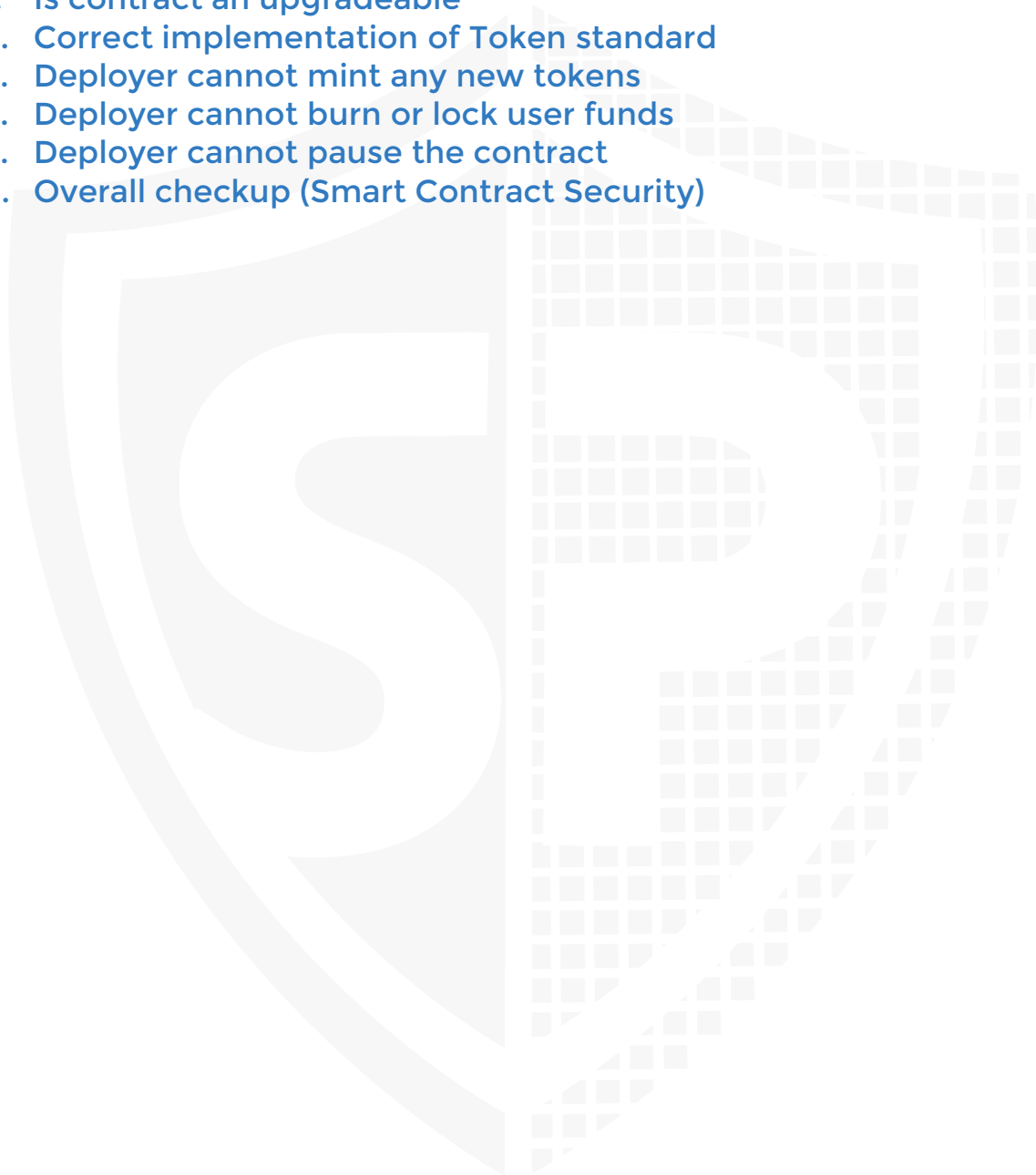


## Scope of Work/Verify Claims

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. Overall checkup (Smart Contract Security)



## Is contract an upgradeable

Name	
Is contract an upgradeable?	No



## 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

addManager
cancelUserVes...
createUserVes...
createUserVes...
createVesting...
createVesting...
removeManager
togglePause
withdraw
withdrawAll



## 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 cannot lock			
Deployer cannot burn			



Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause			



## 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






◆	<Constructor>
◆	createVestingPools
Ⓜ	onlyManager
◆	createVestingPool
Ⓜ	onlyManager
◆	createUserVesting
Ⓜ	onlyManager
◆	createUserVestings
Ⓜ	onlyManager
◆	cancelUserVesting
Ⓜ	onlyManager
◆	withdraw
Ⓜ	whenNotPaused
◆	withdrawAll
Ⓜ	whenNotPaused
◆	togglePause
Ⓜ	onlyManager

## Comments:

- The Manager of the contract can lock the user funds by pausing the contract.
- The manager can cancel vesting in which the cancel is not restricted and that too will be set by the Manager at the time of vesting creation

# Source Units in Scope

v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/Vesting.sol	1	————	354	305	265	10	122	————
	contracts/Managable.sol	1	————	32	32	24	1	13	————
	contracts/libraries/BP.sol	1	————	6	6	4	1	2	————
	contracts/libraries/PercentageVestingLibrary.sol	1	————	102	91	74	10	17	————
	<b>Totals</b>	<b>4</b>	————	<b>494</b>	<b>434</b>	<b>367</b>	<b>22</b>	<b>154</b>	————

## 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	Main	Missing Validation checks	196,178	These functions should check that the parameters passed into them should not be equal and the values must be according to the business logic. For example, vesting duration should not be less than cliff duration, etc.
#2	Main	Floating Pragma	4	The current pragma Solidity directive is " <code>^0.8.0</code> ". Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

## Informational issues

Issue	File	Type	Line	Description
#1	Main	Uninitialized Local Variables	322	Make sure to initialize all the variables.
#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.

13. August, 2022:

- There is still an owner (Owner still has not renounced ownership)
- Read the whole report and modifiers section for more information.



## SWC Attacks

ID	Title	Relationships	Status
<a href="#">SWC-1136</a>	Unencrypted Private Data On-Chain	<a href="#">CWE-767: Access to Critical Private Variable via Public Method</a>	PASSED
<a href="#">SWC-1135</a>	Code With No Effects	<a href="#">CWE-1164: Irrelevant Code</a>	PASSED
<a href="#">SWC-1134</a>	Message call with hardcoded gas amount	<a href="#">CWE-655: Improper Initialization</a>	PASSED
<a href="#">SWC-1133</a>	Hash Collisions With Multiple Variable Length Arguments	<a href="#">CWE-294: Authentication Bypass by Capture-replay</a>	PASSED
<a href="#">SWC-1132</a>	Unexpected Ether balance	<a href="#">CWE-667: Improper Locking</a>	PASSED
<a href="#">SWC-1131</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	PASSED

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

<u>1</u> <u>2</u> <u>4</u>	Storage Location		
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>3</u>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>2</u>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>1</u>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>0</u>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>1</u> <u>1</u> <u>9</u>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	PASSED

<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 8	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 7	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 6	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	NOT PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 5	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 4	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 3	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	PASSED

<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 2	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 1	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 1 0	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 0 9	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 0 8	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	PASSED
<a href="#">S</a> <a href="#">W</a> <a href="#">C</a> : 1 0 7	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	PASSED

<a href="#">SWC-1106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	PASSED
<a href="#">SWC-1105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	PASSED
<a href="#">SWC-1104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	PASSED
<a href="#">SWC-1103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	NOT PASSED
<a href="#">SWC-1102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	PASSED
<a href="#">SWC-1101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	PASSED

SWC-10101	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	PASSED
-----------	-----------------------------------	---	--------





[SolidProof.io](https://solidproof.io)



[@solidproof\\_io](https://t.me/solidproof_io)

Solid  
Proofed

**Blockchain Security | Smart Contract Audits | KYC**

  
MADE IN GERMANY