SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

WaveSignal

Audit

Security Assessment
16. November, 2022

For

WaveSignal

SolidProof_io

@solidproof_io

Disclaimer 3
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 23
Source Units in Scope 26
Critical issues 27
High issues 27
Medium issues 27
Low issues 27
Informational issues 28
Audit Comments 29
SWC Attacks 30

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	16. November 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://www.wavesignal.finance/

Telegram
https://t.me/WaveSignalFinance

Twitter
https://twitter.com/wave_signal

Description

TBA

Project Engagement

During the 14th of November 2022, **WaveSignal Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

WaveSignal

Contract Link

v1.0

- Github
    - https://github.com/wavesignal86/smart-contracts
    - Commit: 7815275

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 aspossible.
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.

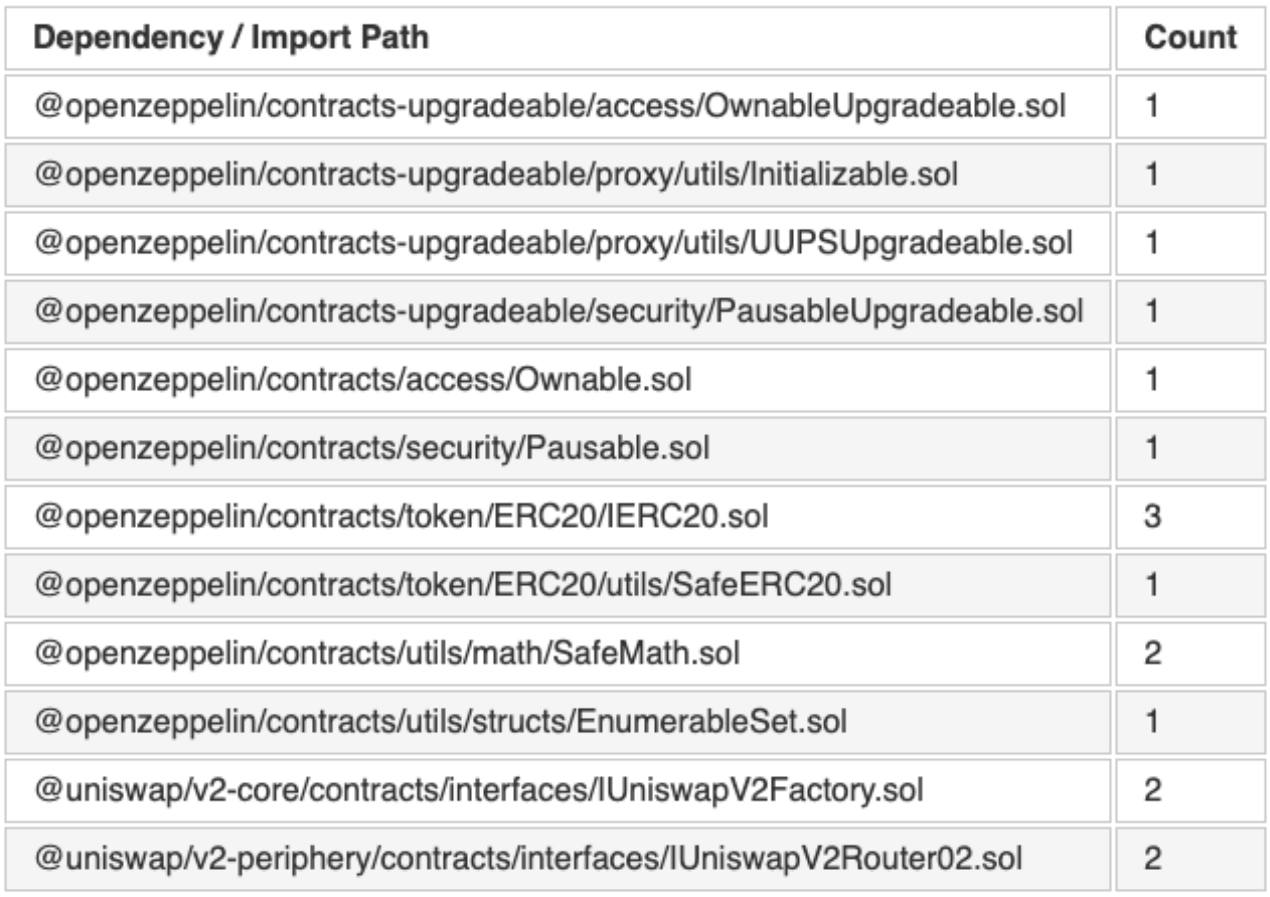
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program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



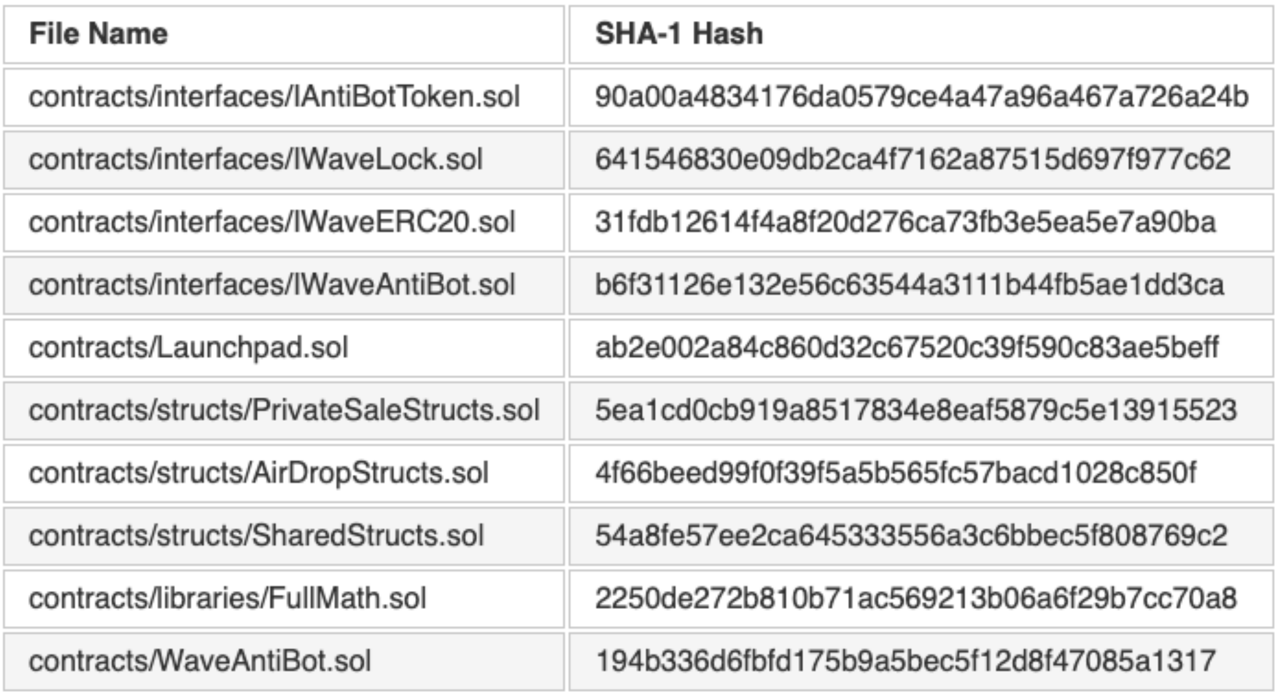
Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	1
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	1
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/security/Pausable.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	3
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	2
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	1
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	2
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	2

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

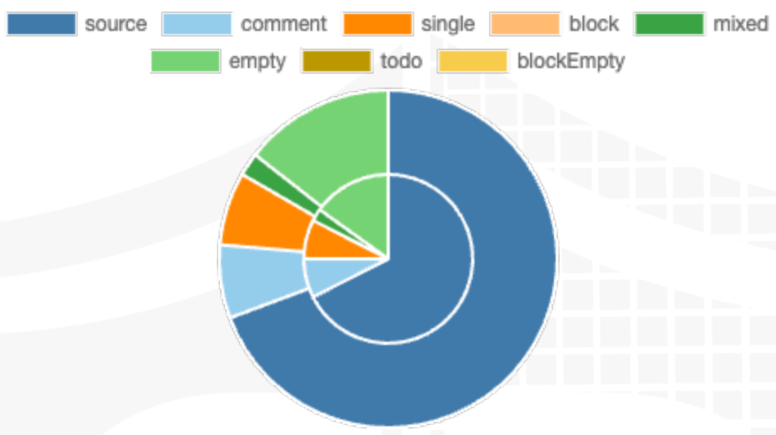


File Name	SHA-1 Hash
contracts/interfaces/IAntiBotToken.sol	90a00a4834176da0579ce4a47a96a467a726a24b
contracts/interfaces/IWaveLock.sol	641546830e09db2ca4f7162a87515d697f977c62
contracts/interfaces/IWaveERC20.sol	31fdb12614f4a8f20d276ca73fb3e5ea5e7a90ba
contracts/interfaces/IWaveAntiBot.sol	b6f31126e132e56c63544a3111b44fb5ae1dd3ca
contracts/Launchpad.sol	ab2e002a84c860d32c67520c39f590c83ae5beff
contracts/structs/PrivateSaleStructs.sol	5ea1cd0cb919a8517834e8eaf5879c5e13915523
contracts/structs/AirDropStructs.sol	4f66beed99f0f39f5a5b565fc57bacd1028c850f
contracts/structs/SharedStructs.sol	54a8fe57ee2ca645333556a3c6bbec5f808769c2
contracts/libraries/FullMath.sol	2250de272b810b71ac569213b06a6f29b7cc70a8
contracts/WaveAntiBot.sol	194b336d6fbfd175b9a5bec5f12d8f47085a1317

Metrics

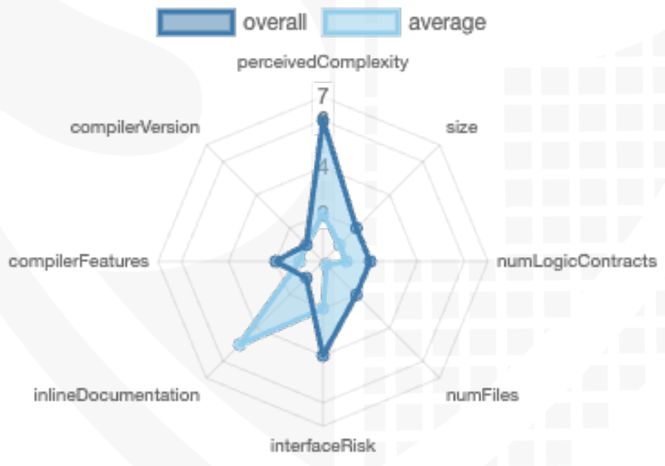
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	4	4	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	47	2

Version	External	Internal	Private	Pure	View
1.0	33	43	1	4	17

State Variables

Version	Total	Public
1.0	52	50

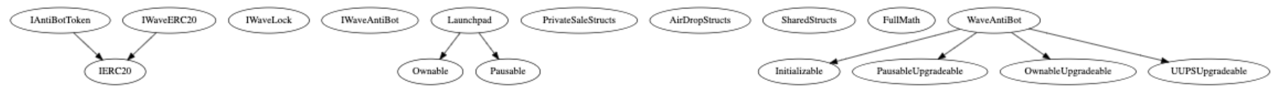
Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.8.4` `^0.8.2` `>=0.4.0`		`yes`	`yes` (8 asm blocks)	

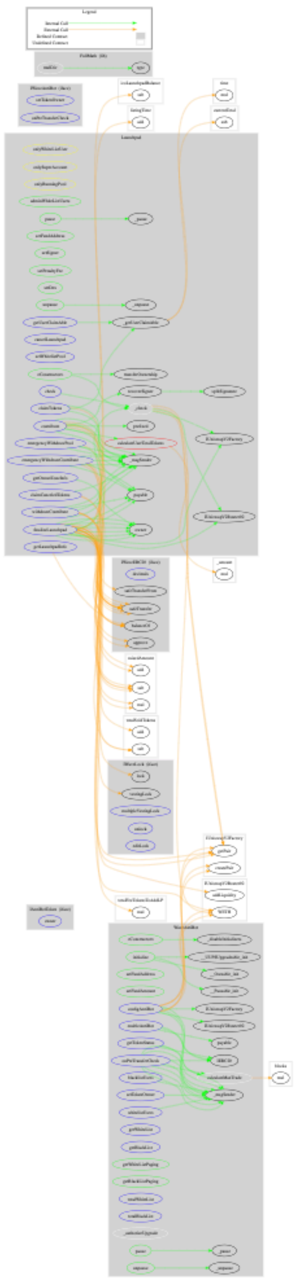
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
1.0	yes			yes	yes	

Inheritance Graph

v1.0



CallGraph
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. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Deployer cannot set fees
6. Deployer cannot blacklist/antisnipe addresses
7. Overall checkup (Smart Contract Security)

Is contract an upgradeable

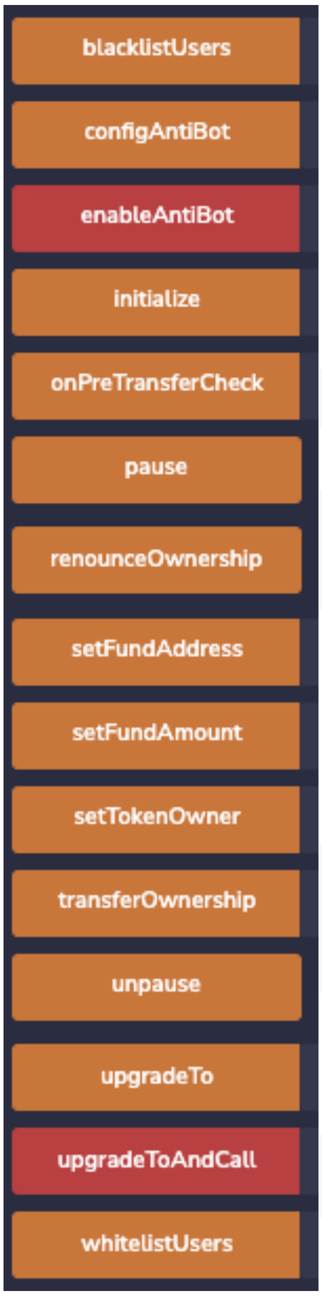
Name	
Is contract an upgradeable?	Yes

Comments:

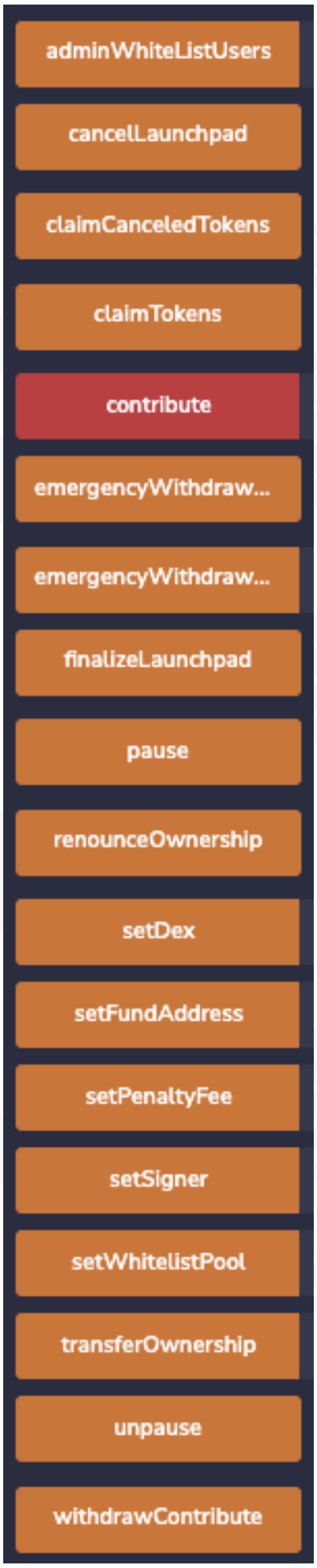
v1.0

- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
    - Be aware of this and do your own research for the contract which is the contract pointing to

Write functions of contract v1.0



blacklistUsers
configAntiBot
enableAntiBot
initialize
onPreTransferCheck
pause
renounceOwnership
setFundAddress
setFundAmount
setTokenOwner
transferOwnership
unpause
upgradeTo
upgradeToAndCall
whitelistUsers



adminWhiteListUsers
cancelLaunchpad
claimCanceledTokens
claimTokens
contribute
emergencyWithdraw...
emergencyWithdraw...
finalizeLaunchpad
pause
renounceOwnership
setDex
setFundAddress
setPenaltyFee
setSigner
setWhitelistPool
transferOwnership
unpause
withdrawContribute

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	–	–	–

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock	✓	✓	✗
Deployer cannot burn	✓	✓	✗

Comments:

v1.0

- WaveAntiBot
    - Owner can lock user funds by
        - blacklisting addresses
- Launchpad
    - Owner can set penaltyFee without any limitation. This can lock the emergencyWithdrawContribute function

- While finalizing launchpad the totalRefundOrBurnTokens will be burned if the pool type is set to 0

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	✓	✓	✗

Comments:

v1.0

- Owner can
    - pause contract
    - lock user funds by pausing the contract. You are not able to claim tokens anymore

Deployer cannot set fees

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

Comments:

v1.0

- Fees can be set without any limitations

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	✓	✓	✗

Comments:

v1.0

- Addresses can be blacklisted by the owner

Overall checkup (Smart Contract Security)

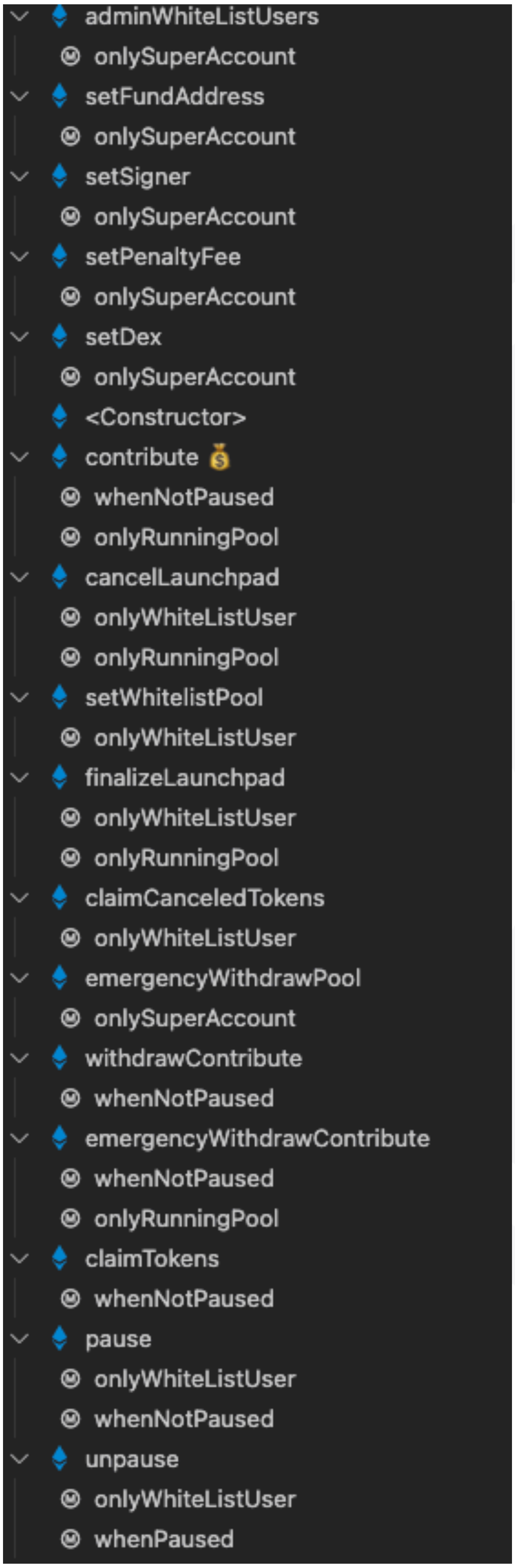
Tested	Verified
✓	✓

Legend

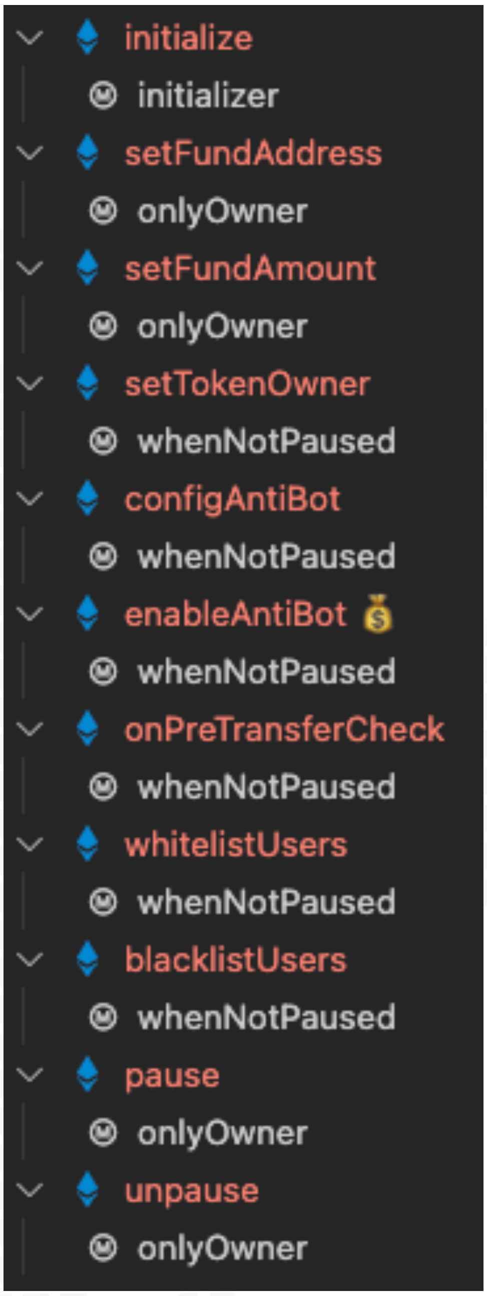
Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	–

Modifiers and public functions

v1.0



```
adminWhiteListUsers
    onlySuperAccount
setFundAddress
    onlySuperAccount
setSigner
    onlySuperAccount
setPenaltyFee
    onlySuperAccount
setDex
    onlySuperAccount
<Constructor>
contribute 💰
    whenNotPaused
    onlyRunningPool
cancelLaunchpad
    onlyWhiteListUser
    onlyRunningPool
setWhitelistPool
    onlyWhiteListUser
finalizeLaunchpad
    onlyWhiteListUser
    onlyRunningPool
claimCanceledTokens
    onlyWhiteListUser
emergencyWithdrawPool
    onlySuperAccount
withdrawContribute
    whenNotPaused
emergencyWithdrawContribute
    whenNotPaused
    onlyRunningPool
claimTokens
    whenNotPaused
pause
    onlyWhiteListUser
    whenNotPaused
unpause
    onlyWhiteListUser
    whenPaused
```



```
initialize
    initializer
setFundAddress
    onlyOwner
setFundAmount
    onlyOwner
setTokenOwner
    whenNotPaused
configAntiBot
    whenNotPaused
enableAntiBot 💰
    whenNotPaused
onPreTransferCheck
    whenNotPaused
whitelistUsers
    whenNotPaused
blacklistUsers
    whenNotPaused
pause
    onlyOwner
unpause
    onlyOwner
```

Note: Functions from imported libraries will not be listed here

Comments

- *Deployer can set following state variables without any limitations*
  - Launchpad
    - penaltyFee
    - holdingTokenAmount
  - WaveAntiBot
    - fundAmount
  -
- *Deployer can enable/disable following state variables*
  - Launchpad
    - whiteListUsers
    - _paused
  - WaveAntiBot
    - whitelistedUsers
      - Only the owner of antiBotInfo can set it
    - blacklistedUsers
      - Only the owner of antiBotInfo can set it
    - _paused

- *Deployer can set following addresses*
  - Launchpad
    - fundAddress
    - signer
    - factoryAddress
    - routerAddress
    - holdingToken
  - WaveAntiBot
    - fundAddress
    - antiBotInfo.owner
      - Anyone can set antibotInfo owner

- *Existing Modifiers*
  - Launchpad
    - onlyWhiteListUser
    - onlySuperAccount
    - onlyRunningPool

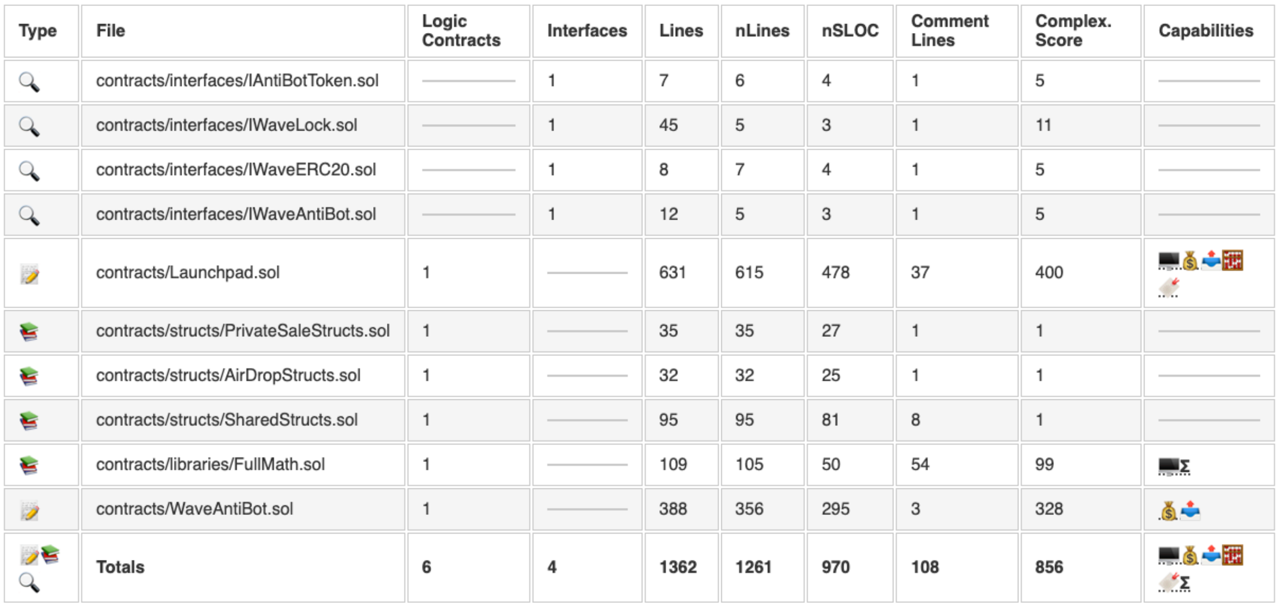
- There are several authorities which are authorized to call some functions, that means, if the owner is renounced, another address is still authorized to call functions
  - Be aware of this
- OnlyWhitelistuser can

    - Cancel launchpad and set the state to 3
- Anyone can set antiBotInfo when he’s set as the owner

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/interfaces/IAntiBotToken.sol	—	1	7	6	4	1	5	—
🔍	contracts/interfaces/IWaveLock.sol	—	1	45	5	3	1	11	—
🔍	contracts/interfaces/IWaveERC20.sol	—	1	8	7	4	1	5	—
🔍	contracts/interfaces/IWaveAntiBot.sol	—	1	12	5	3	1	5	—
📝	contracts/Launchpad.sol	1	—	631	615	478	37	400	🖥💰📤🧮 🔖
📚	contracts/structs/PrivateSaleStructs.sol	1	—	35	35	27	1	1	—
📚	contracts/structs/AirDropStructs.sol	1	—	32	32	25	1	1	—
📚	contracts/structs/SharedStructs.sol	1	—	95	95	81	8	1	—
📚	contracts/libraries/FullMath.sol	1	—	109	105	50	54	99	🖥Σ
📝	contracts/WaveAntiBot.sol	1	—	388	356	295	3	328	💰📤
📝📚 🔍	**Totals**	**6**	**4**	**1362**	**1261**	**970**	**108**	**856**	🖥💰📤🧮 🔖Σ

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised 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

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	Main	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	Launch pad	A floating pragma is set	2	The current pragma Solidity directive is „"^0."".
#3	WaveAn tiBot	A floating pragma is set	2	The current pragma Solidity directive is „"^0."".
#4	Launch pad	Missing Zero Address Validation (missing-zero-check)	151, 138, 142	Check that the address is not zero
#5	WaveAn tiBot	Local variables shadowing	92	Rename the local variables that shadow another component Here the "owner" variable is shadowing the OwnableUpgradeable.owner( ) function
#6	WaveAn tiBot	Missing Events Arithmetic	89	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	Launch pad	State variables that could be declared constant (constable-states)	116	Add the `constant` attributes to state variables that never change
#2	WaveAn tiBot	Array declaration	299, 286, 324, 350	There should be no whitespace between type of the variable and opening swore brackets

#3	Launch pad	Check PoolType	179	We recommend you to check the poolType in the constructor because nobody is able to change the type of the pool
				
				Also we recommend you to implement a struct for the PoolType like
				
				struct PoolType {
				Burn,
				Refund
				}
				
				and use this instead of magic numbers (0 and 1).
				
				The same for:
				- whitelistPool in L72
				- State in L92

#4	Launch pad	Unnecessary if condition	392	Remove the if statement because it is already checked before with the require statement
#5	WaveAn tiBot	Initialize function	70	Everybody is able to call the initialize function. Make sure to call it automatically while deploying in your script. Otherwise everybody can call it.
#6	Launch pad	Visibility should come first	70	“Public” should come before other modifiers
#7	WaveAn tiBot	Unused interface	17	Remove or use the interface

#8	WaveAntiBot	Unnecessary library	13	Above pragma version 0.8.x overflow/underflow will be checked from the version by default. Replace all “SafeMath” libraries with raw mathematical operations.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/latest/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.

16. November 2022:

- Following contracts were not provided to solidproof. DYOR here.
    - WaveERC20
    - WaveLock
- WhitelistUsers can claim whole balance of icoToken after canceling
- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
- Read 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-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
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-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	**PASSED**
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	**NOT 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

SolidProof_io

@solidproof_io

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY