SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

Kelp

Audit

Security Assessment
11. May, 2023

For

kelp

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 24
Source Units in Scope 26
Critical issues 27
High issues 27
Medium issues 27
Low issues 27
Informational issues 27
Audit Comments 28
SWC Attacks 29

Disclaimer

Version	Date	Description
1.0	10. May 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://kelp.finance

Telegram
https://t.me/kelpfinance

Twitter
https://twitter.com/KelpFinance

Facebook
https://www.facebook.com/kelpfinance

Youtube
https://www.youtube.com/@kelpfinance

Description

TBA

Project Engagement

During the 9th of May 2023, **Kelp 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

kelp

Contract Link

v1.0

- Github
    - https://github.com/DioneProtocol/Governance-sc
    - Commit: 75bd04a

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.

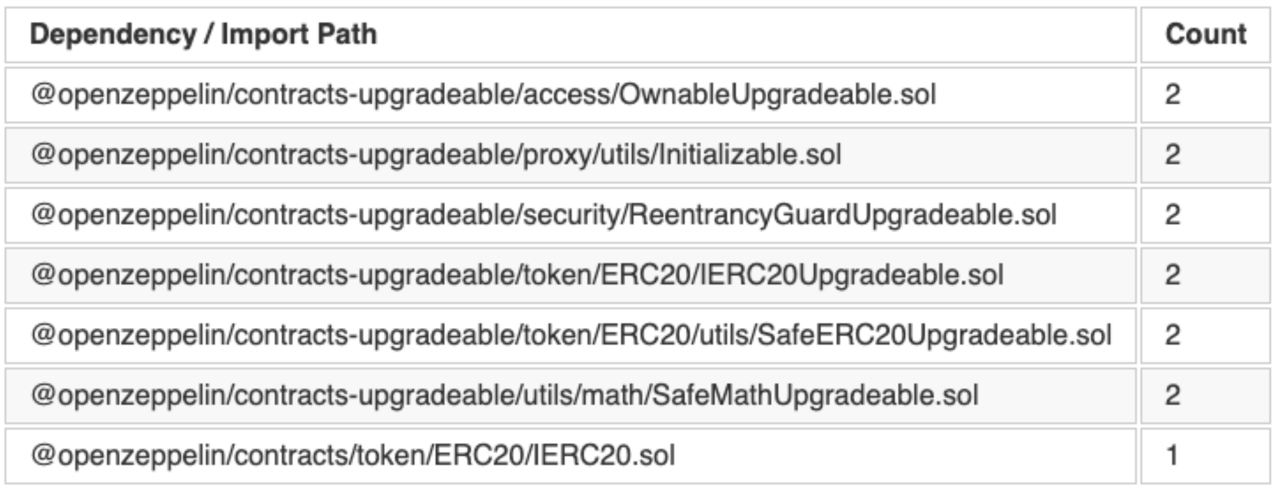
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program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



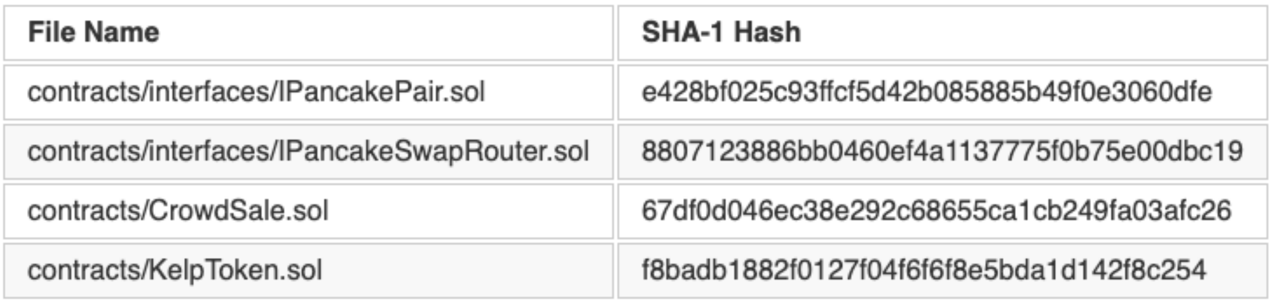
Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	2
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	2
@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol	2
@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol	2
@openzeppelin/contracts/token/ERC20/IERC20.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

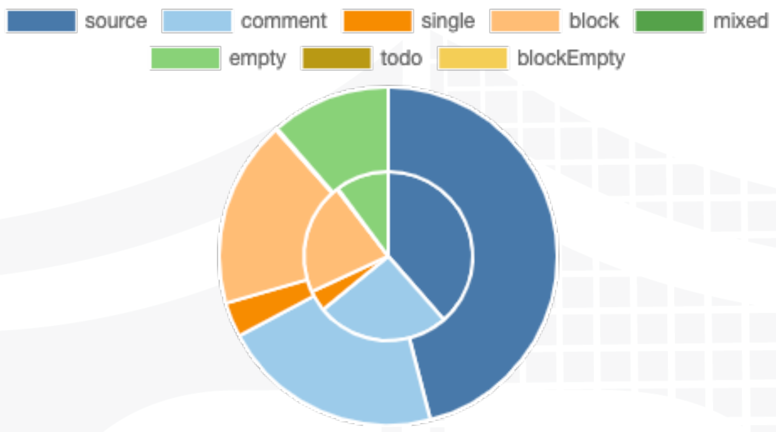


File Name	SHA-1 Hash
contracts/interfaces/IPancakePair.sol	e428bf025c93ffcf5d42b085885b49f0e3060dfe
contracts/interfaces/IPancakeSwapRouter.sol	8807123886bb0460ef4a1137775f0b75e00dbc19
contracts/CrowdSale.sol	67df0d046ec38e292c68655ca1cb249fa03afc26
contracts/KelpToken.sol	f8badb1882f0127f04f6f6f8e5bda1d142f8c254

Metrics

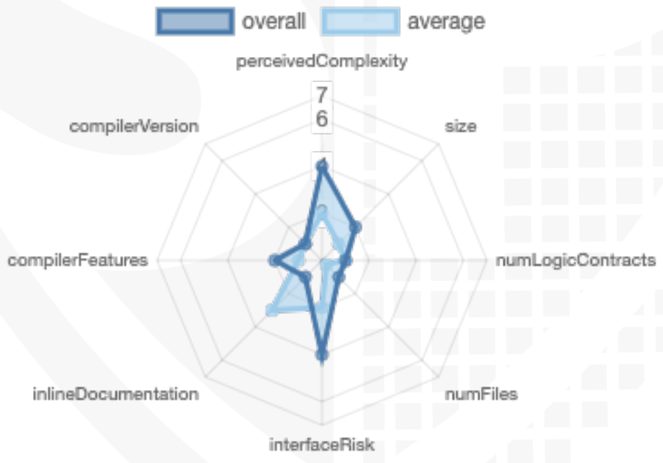
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	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.

Version	Public	Payable
1.0	50	1

Version	External	Internal	Private	Pure	View
1.0	42	30	0	5	18

State Variables

Version	Total	Public
1.0	20	19

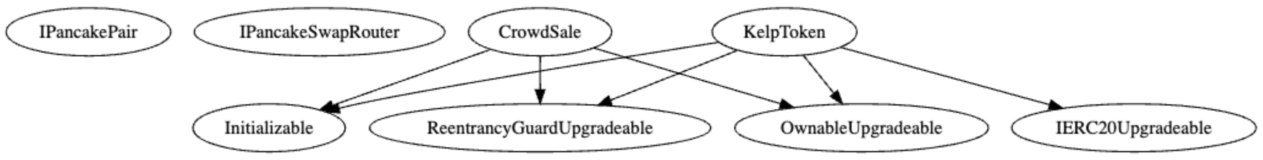
Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.8.12`		`yes`		

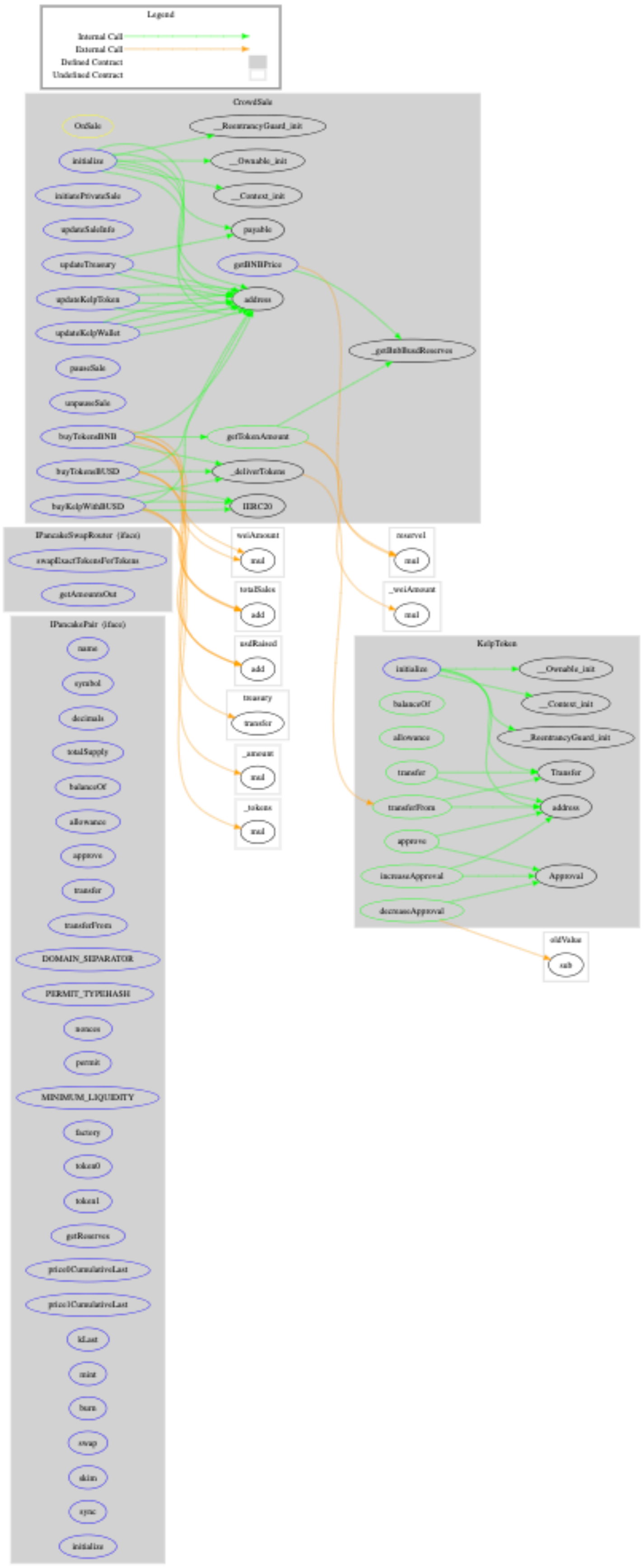
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2

1.0	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. 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 cannot set fees
7. Deployer cannot blacklist/antisnipe addresses
8. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	Yes

Comments:

v1.0

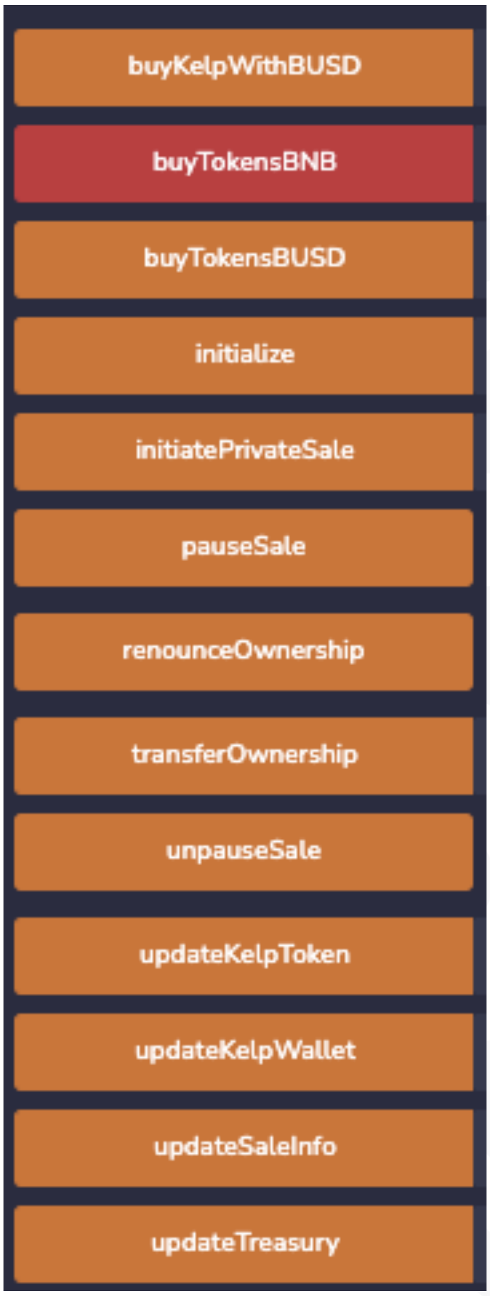
- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
    - Be aware of this and do your own research for the contract which is the contract pointing to

Correct implementation of Token standard

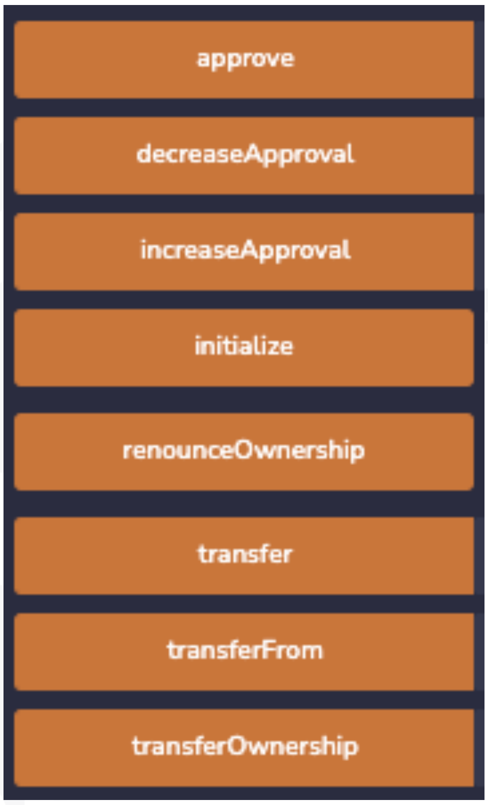
ERC20				
Function	**Description**	**Exist**	**Tested**	**Verified**
TotalSupply	Provides information about the total token supply	✓	✓	✓
BalanceOf	Provides account balance of the owner's account	✓	✓	✓
Transfer	Executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	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 contract v1.0

CrowdSale.sol


buyKelpWithBUSD
buyTokensBNB
buyTokensBUSD
initialize
initiatePrivateSale
pauseSale
renounceOwnership
transferOwnership
unpauseSale
updateKelpToken
updateKelpWallet
updateSaleInfo
updateTreasury

KelpToken.sol


approve
decreaseApproval
increaseApproval
initialize
renounceOwnership
transfer
transferFrom
transferOwnership

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	80000000000		

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Owner can lock by setting the kelp address to an address that doesn’t support the transferFrom function. Additionally to it the owner can also decrease the allowance of the kelp wallet.

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	✓	✓	✗

Comments:

v1.0

- Owner can pause CrowdSale contract as long as the the current time is between the range of startTime an endTime (14 days)

Deployer cannot set fees

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

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	–	–	–

Overall checkup (Smart Contract Security)

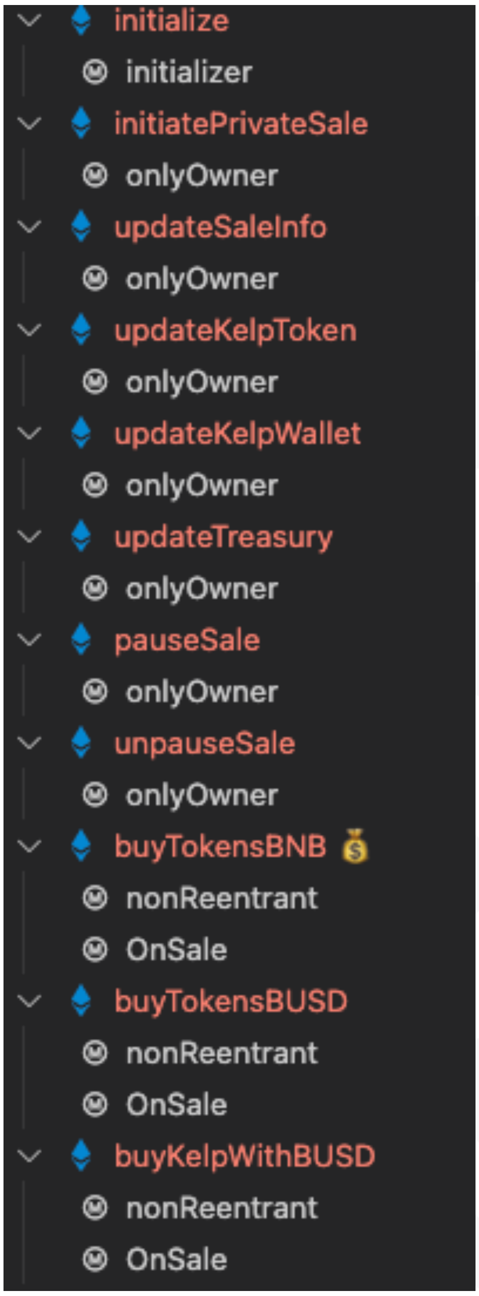
Tested	Verified
✓	✓

Legend

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

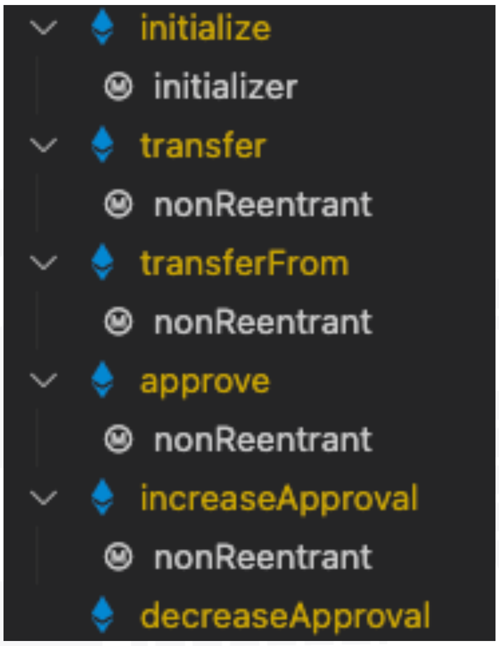
Modifiers and public functions
v1.0

CrowdSale



```
initialize
    initializer
initiatePrivateSale
    onlyOwner
updateSaleInfo
    onlyOwner
updateKelpToken
    onlyOwner
updateKelpWallet
    onlyOwner
updateTreasury
    onlyOwner
pauseSale
    onlyOwner
unpauseSale
    onlyOwner
buyTokensBNB 💰
    nonReentrant
    OnSale
buyTokensBUSD
    nonReentrant
    OnSale
buyKelpWithBUSD
    nonReentrant
    OnSale
```

KelpToken



```
initialize
    initializer
transfer
    nonReentrant
transferFrom
    nonReentrant
approve
    nonReentrant
increaseApproval
    nonReentrant
decreaseApproval
```

Comments

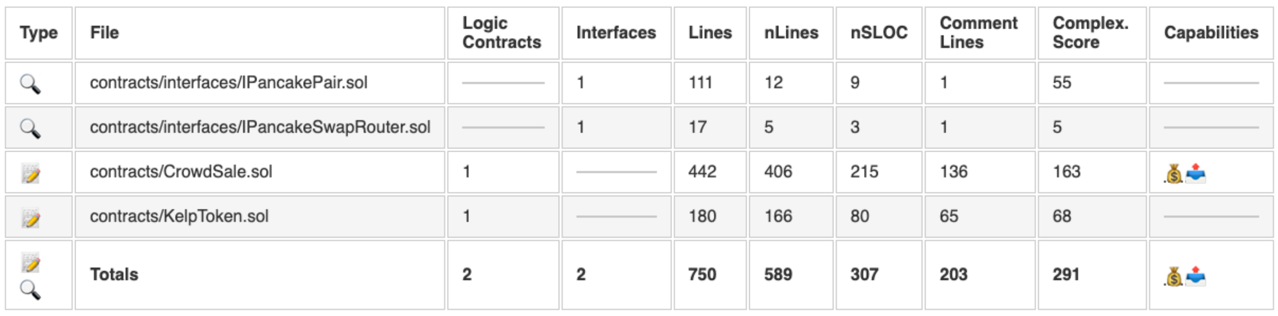
- In the CrowSale contract the owner has following permissions:
    - initiatePrivateSale
        - Initiate private sale as long as the _startTime is above the current timestamp. The owner is able to call this function all the time. If it is the purposes that the owner can start sales again it is recommended to prevent the call when a sale is running.
    - updateSaleInfo

    - Update the initiated infos as long as the sale is not started already. The owner is able to set the saleLimit and tokenPrice to an arbitrary value in the uint256 range.
- updateKelpToken
    - Updating the kelp token address. If the kelp token address is an address where the kelpWallet has no funds in it the crowdsale contract will not work properly
- updateKelpWallet
    - Updating the kelp wallet from where the tokens will be send
- updateTreasury
    - Updating the treasury where the native tokens will be sen after buying tokens with bnb and busd. This can be also a EOA (External owned address) who can get out the funds.
- Pause-/unpauseSale
    - Pause and unpause the contract. pauseSale can only be called if the current timestamp is between the range of start and end time

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/IPancakePair.sol	—	1	111	12	9	1	55	—
🔍	contracts/interfaces/IPancakeSwapRouter.sol	—	1	17	5	3	1	5	—
📝	contracts/CrowdSale.sol	1	—	442	406	215	136	163	💰📥
📝	contracts/KelpToken.sol	1	—	180	166	80	65	68	—
📝🔍	**Totals**	**2**	**2**	**750**	**589**	**307**	**203**	**291**	💰📥

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	All	A floating pragma is set	2	The current pragma Solidity directive is „“^0.8.12””.
#2	Main	Local variables shadowing	64, 54	Rename the local variables that shadow another component. It is recommended to move the underscore to the end of the “owner” instead of the beginning.

Informational issues

Issue	File	Type	Line	Description
#1	CrowdSale	Wrong comment	353	The caller is the one who performs the purchase, not the beneficiary. The beneficiary is only the one who will get the tokens at the end of the purchase.
#2	KelpToken	Naming convention	33, 26-30	Start a private/internal variables/functions with an underscore. Also write variables that are constant with uppercased letters.

#3	Tests	Unavailable function	See description	The tests are still calling functions that are has been removed from the contracts. Adjust your test cases for the current contract functions. Functions called: - crowdSale.updateKelpOwner - crowdSale.updateKelpOwner - crowdSale.pauseUnpauseSale
#4	Tests	Start timestamp	See description	When your contract uses timestamps in the contract (Sale timestamp) it is recommended to start the test cases at a specific time stamp. Otherwise you have to adjust the timestamp in the tests all the time when you try to test your tests on a later time. At this time the test cases will revert with the "can't set startTime in the past" error message.

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.

11. May 2023:

- Anyone is able to buy tokens for other addresses
- 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

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY