SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Pinksale

Audit

Security Assessment

02. June, 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	02. May 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://www.pinkmoon.finance/#/

Telegram
https://t.me/pinkmoonfinance

Twitter
https://twitter.com/pinkmoonfinance

Github
https://github.com/pinkmoonfinance

Medium
https://medium.com/@pinkmoonfinance

Description

PinkSale is a decentralized launchpad that allows users to launch their own token and create their own initial token sale. No coding knowledge is required, just simply navigate through to our terminal and design your own token in just a few clicks. PinkSale offers multiple other features to help you with the overall token launch, such as: Automatic listing of your token on PancakeSwap, UniSwap, ShibaSwap, SushiSwap, KuSwap, QuickSwap and MM Finance, all whilst giving you the ability to lock your LP and and adding an optional vesting period for your tokens.

Project Engagement

During the 1st of June 2022, **PinkMoon Finance 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

Contract Link

v1.0

- Github
  - https://github.com/pinkmoonfinance/pink-lock-contracts-v3
  - Commit: 41c23000144f19d6f21f54274acb48735dcc9fef

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.

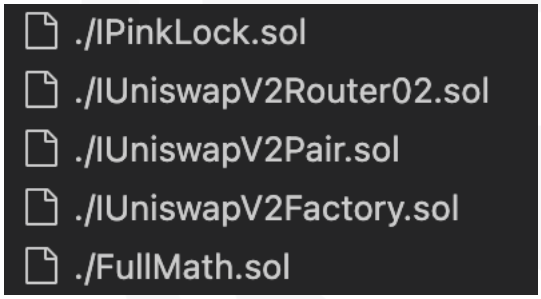
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

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/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1
@openzeppelin/contracts/utils/Address.sol	1
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	1



Tested Contract Files

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

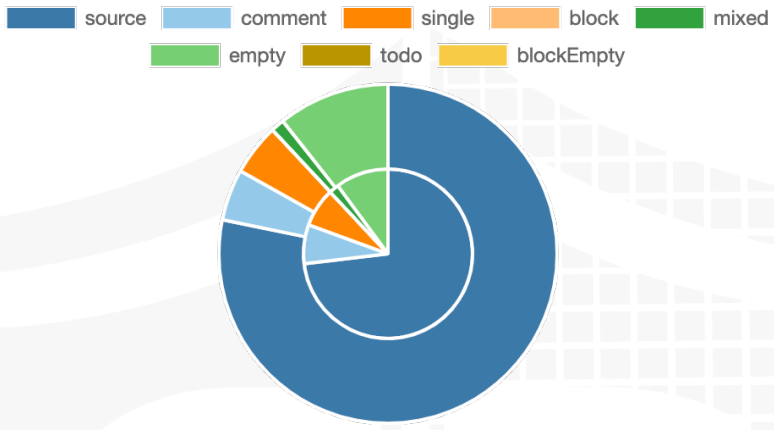
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

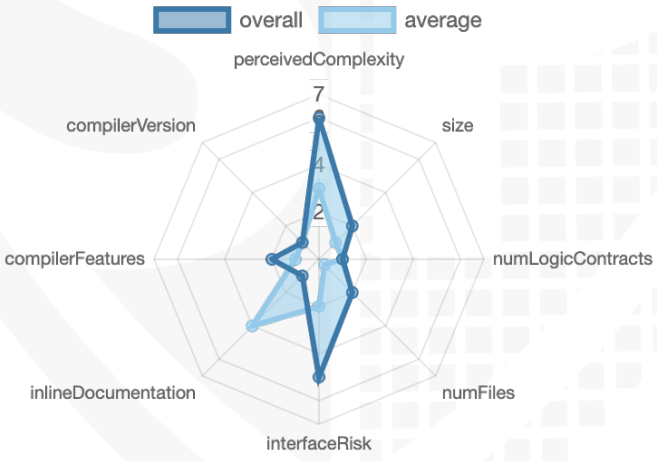
File Name	SHA-1 Hash
contracts/IUniswapV2Pair.sol	d01c1a641b0eefea16170c00919892e194eb0253
contracts/FullMath.sol	352f6d24430fe319329e9ecf66a1f628597d4f8c
contracts/IUniswapV2Factory.sol	f6f535836b25cacb92c0c63a9c06ec933800d5d9
contracts/PinkLock02.sol	a7dda37ab57d7c3d12abe84e3a5ff791bda87283
contracts/IUniswapV2Router02.sol	02ae557581982c9f100a13e0ef50b83e790daaa5
contracts/IPinkLock.sol	9fc63e63f74830ab3c4a976ce29779a50c88a50e

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	1	1	5	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	92	4

Version	External	Internal	Private	Pure	View
1.0	85	57	4	12	43

State Variables

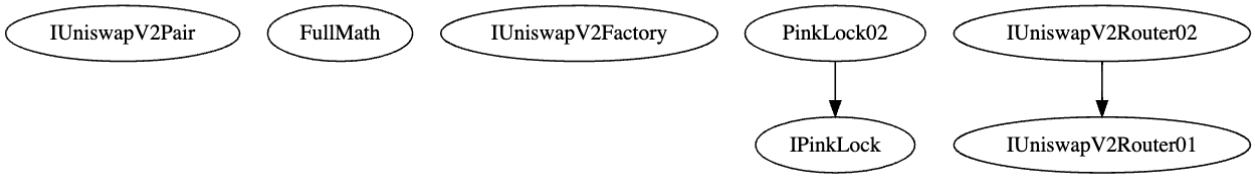
Version	Total	Public
1.0	8	1

Capabilities

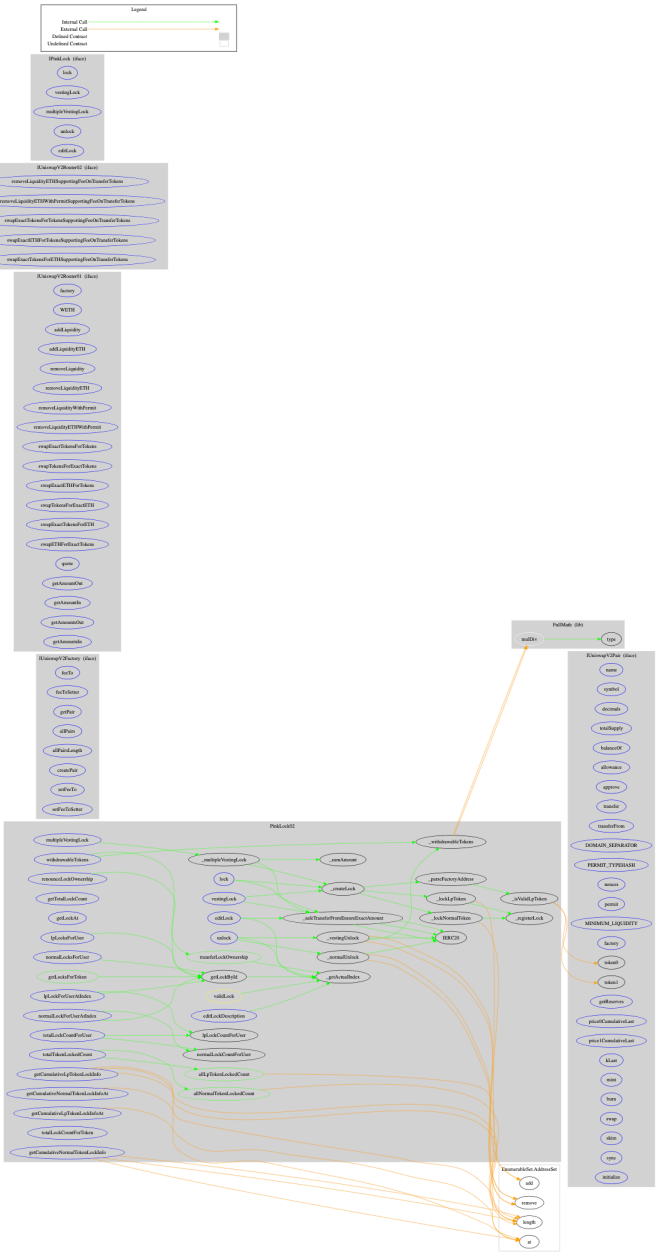
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`=0.8.4` `>=0.4.0`		`yes`	`yes` (7 asm blocks)	

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. Overall checkup (Smart Contract Security)

Write functions of contract v1.0

lock
vestingLock
multipleVestingLock
unlock
editLock
editLockDescription
transferLockOwnership
renounceLockOwnership

Overall checkup (Smart Contract Security)

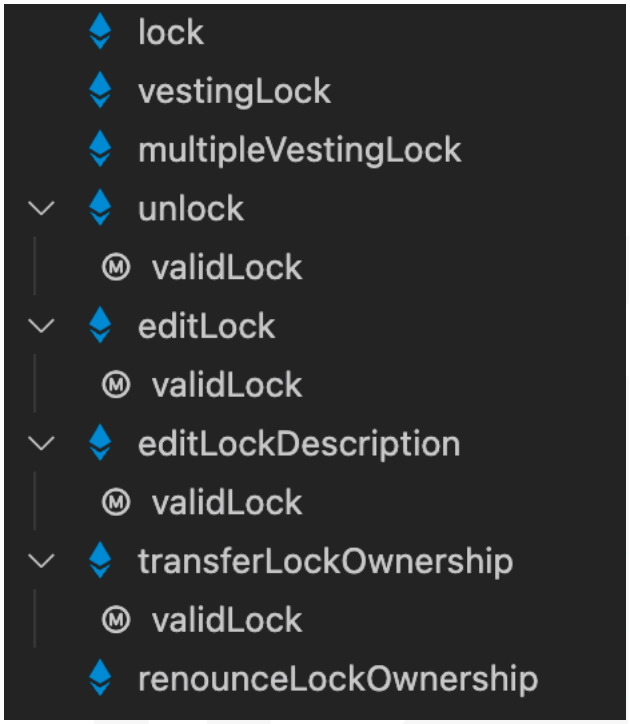
Tested	Verified
✓	✓

Legend

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

Modifiers and public functions

v1.0



Note: Not listed functions are implemented from libraries

Comments

- *Existing Modifiers*
  - validLock

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/IUniswapV2Pair.sol		1	105	8	5	1	55	
📚	contracts/FullMath.sol	1		109	105	50	54	99	💻Σ
🔍	contracts/IUniswapV2Factory.sol		1	32	12	9	1	17	
📝	contracts/PinkLock02.sol	1		878	723	636	15	362	♻️
🔍	contracts/IUniswapV2Router02.sol		2	208	6	4	1	64	💰
🔍	contracts/IPinkLock.sol		1	45	5	3	1	11	
📝📚🔍	**Totals**	**2**	**5**	**1377**	**859**	**707**	**73**	**608**	💻💰♻️Σ

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

No low issues

Informational issues

Issue	File	Type	Line	Description
#1	Main	Misspelling	See description	Change following words: - bips L28, L30 - lastest L42, L141, L142, L183 - transfered L664 Make sure to change it everywhere else as well.
#2	Main	Initialized variable	L852	If you have started to initialize local variables, initialize the others also. Compared to L537, L544

#3	Main	Unnecessary modifier in L86	389, 557, 608, 621	We recommend you to use “_getActualIndex” function instead of the modifier because of duplication. The modifier “validLock” is calling the “_getActualIndex” function which reverts the function or returns the actualIndex. And after the modifier has been executed you are calling again the “_getActualIndex” function to get the Index in every function where the modifier is used. The “_getActualIndex” is reverting already if the passed lockId is smaller than ID_PADDING and “actualIndex” is higher than “_locks” length
#4	Main	Duplicates	Look at the source code	There are many situations where the code is repeating. You can put some logic into functions to get rid of repetitions

Contract testing results

PinkLock02

- ✔ can lock normal token (427ms)
- ✔ can lock Lp token (394ms)
- ✔ can edit lock (463ms)
- ✔ cant edit lock with lower amount (391ms)
- ✔ cant edit lock with lower unlock time (388ms)
- ✔ only owner can edit their lock (376ms)
- ✔ cant edit unlocked lock (411ms)
- ✔ can unlock lock (409ms)
- ✔ cumulative info updated (459ms)
- ✔ Update total lock count for token (417ms)
- ✔ Can get locks for token (454ms)
- ✔ Returns lock count for user (438ms)
- ✔ Returns lock for user at index (482ms)
- ✔ Can edit add new amount (385ms)
- ✔ Not allow using tax token (536ms)
- ✔ Can do vesting lock (374ms)
- ✔ Can unlock vesting lock (466ms)
- ✔ Can unlock vesting lock in one tx (397ms)
- ✔ Revert when supplying invalid lock id (560ms)
- ✔ Cannot do vesting lock given invalid params (421ms)
- ✔ Cant unlock lock if is not the owner (367ms)
- ✔ Cant unlock normal lock if unlock time haven't been passed (363ms)
- ✔ Cant unlock normal lock if already unlocked (390ms)
- ✔ Properly update cumulative amount when unlocked (478ms)
- ✔ Returns correct total lock count (405ms)
- ✔ Returns correct total lock count (428ms)
- ✔ Returns correct lock at specific index (393ms)
- ✔ Returns lp and normal token lock info (830ms)
- ✔ Returns correct all tokens lock count (810ms)
- ✔ Revert if supplying an invalid LP token (527ms)
- ✔ Can renounce lock ownership (392ms)
- ✔ Can do multiple vesting (369ms)

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.

02. June 2022:

- 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	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	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