SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

IceCreamSwap

Decentralized exchange

Audit

Security Assessment

13. June, 2023

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	12. June 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Bitgert
Core
XDC
Binance smart chain (only bridge deployed for now)
Dogechain
Fuse

Website

https://icecreamswap.com/?chainId=1116

Telegram

https://t.me/Icecreamswap_com

Twitter

https://twitter.com/icecream_swap

Description

Trade, Earn, Bridge and Launch on CORE, XDC, Binance smart chain (BSC), Bitgert (Brise), Shardeum, Dogechain, Doken and Fuse with our decentralized smart contracts.

Project Engagement

During the 26th of May 2023, **IceCreamSwap 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

- Dex
  - https://github.com/IceCreamSwapCom/IceCreamSwap-smart-contracts/tree/master/projects/dex
  - Commit: 77bc187fe1d714695d434d29cdc08fa29c9f0811

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:
-

Note:

- **The core folder is a fork of the Uniswap V2 project. The only changes are the types. The types has been changed to an explicit type from “uint” to “uint256”**
- **The same appears to the periphery project folder**

Tested Contract Files

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

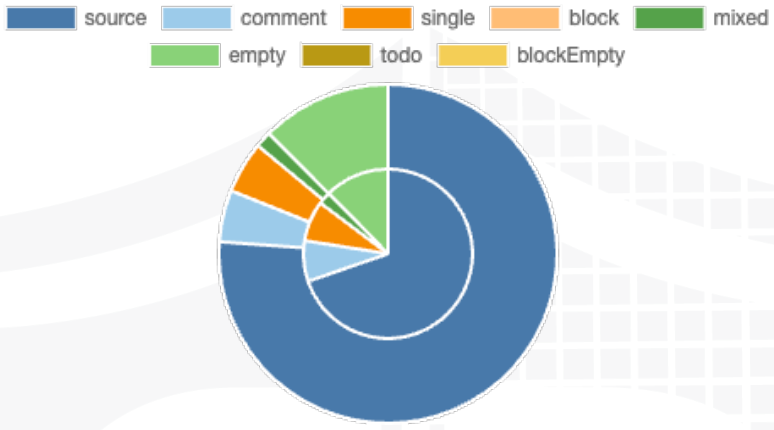
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

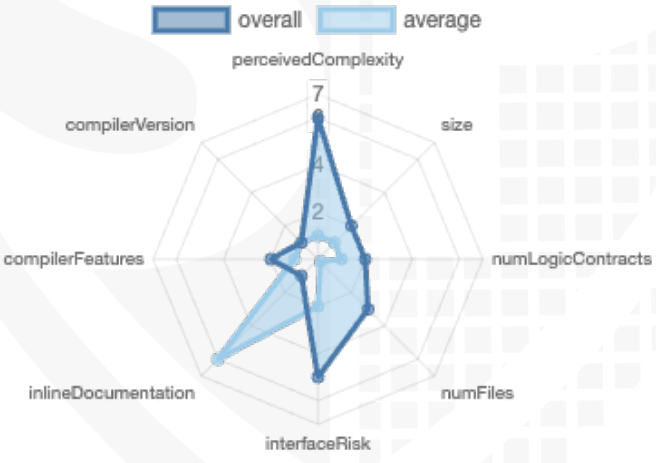
File Name	SHA-1 Hash
contracts/periphery/interfaces/IUniswapV2Migrator.sol	fb583a0edfce4f494f48189f890f89ebab6ee529
contracts/periphery/interfaces/IWETH.sol	f38b78bc02631c83b321016f4cb723aad1ff525b
contracts/periphery/interfaces/IUniswapV2Router02.sol	f42cf72fc78ccb2a4b1887b0ead252fd388a4cdc
contracts/periphery/interfaces/IUniswapV2Router01.sol	3c84972dd56ae191e6f92eb318c6e86269a9628c
contracts/periphery/interfaces/IERC20.sol	201d395a2852148aeef128325254be2c4778cac6
contracts/periphery/UniswapV2Router02.sol	c105ea3b651f6878a59b500c70d9405a16fa8b1a
contracts/periphery/libraries/UniswapV2LiquidityMathLibrary.sol	07af2edb18d70a973a40b4342607db8607780c90
contracts/periphery/libraries/SafeMath.sol	5abf8348da982d3dabcc96986d2e4d067dd27b81
contracts/periphery/libraries/UniswapV2Library.sol	fcd5da11067f3c4c846d797e336e51b6d8e81c61
contracts/periphery/libraries/UniswapV2OracleLibrary.sol	00322008ce7c47c8a2cccafc0aa75928b8faa081
contracts/core/interfaces/IUniswapV2Callee.sol	54e7af50c5f461db1043f05a30b50e79042ad740
contracts/core/interfaces/IUniswapV2Pair.sol	3d66788733272c9dc46c8ba31092b38d2ebce2f0
contracts/core/interfaces/IUniswapV2Factory.sol	8ad0844a0d3c9d73606280ad4925c87842a5c22e
contracts/core/interfaces/IUniswapV2ERC20.sol	97feb910035203b101fd8688b2247cec813dbeb4
contracts/core/interfaces/IERC20.sol	201d395a2852148aeef128325254be2c4778cac6
contracts/core/UniswapV2Pair.sol	a9367dff07e9f96b353aa665d3f39a6c05f5ecdd
contracts/core/libraries/Math.sol	85d723f1f7c03aa6eda8ee62192fa587e08477bf
contracts/core/libraries/UQ112x112.sol	5c0f96357914f9f80b6d616b79ece099d5f91ec4
contracts/core/libraries/SafeMath.sol	cd5ec30d1cf82a104be0ddb83fffe6a65392529a
contracts/core/UniswapV2ERC20.sol	44c2b2f01a618964253f4911d169edf68d31b5a5
contracts/core/UniswapV2Factory.sol	c758926c3b36bfcff64d6e415378a6847896adf1

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	4	7	10	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	137	10

Version	External	Internal	Private	Pure	View
1.0	124	110	5	34	48

State Variables

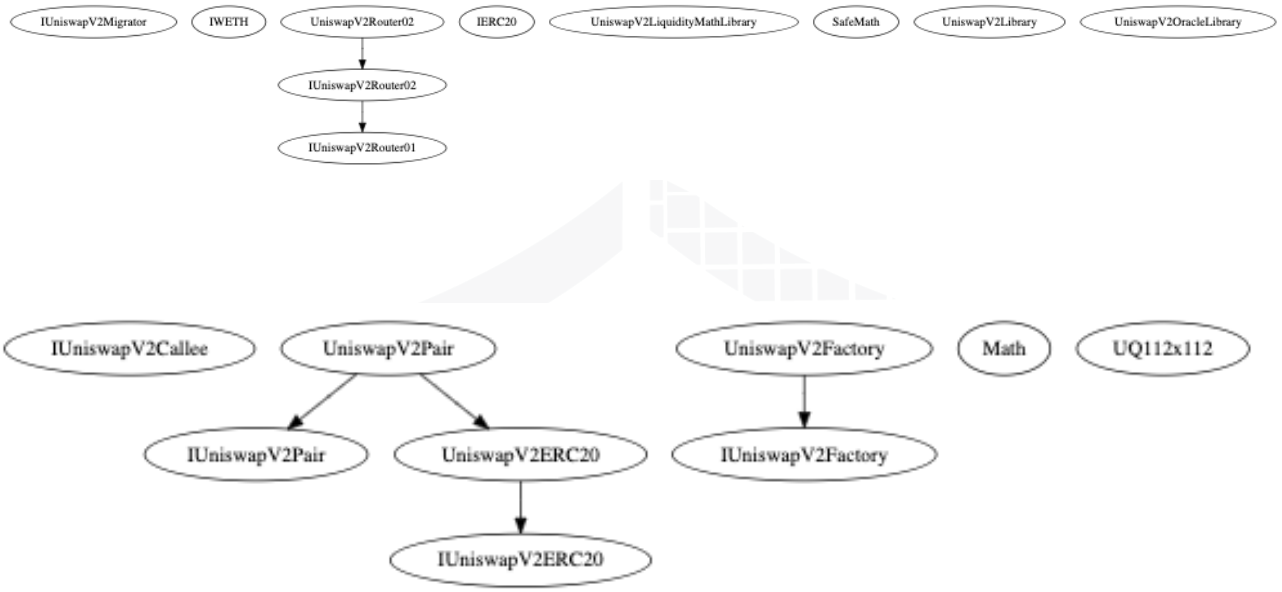
Version	Total	Public
1.0	29	23

Capabilities

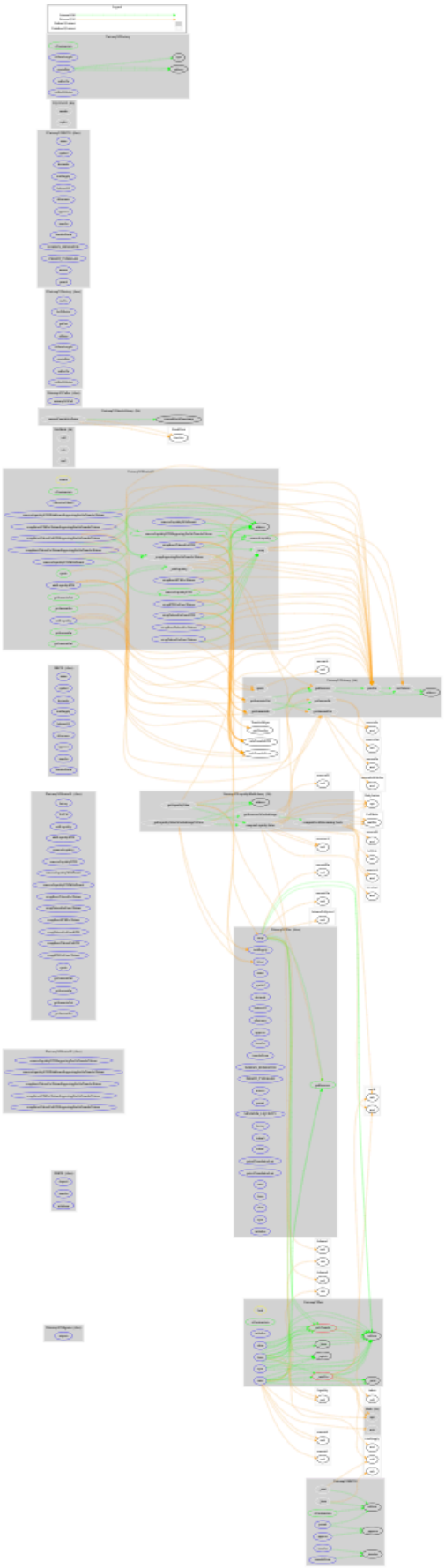
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`>=0.5.0` `>=0.6.2` `=0.6.6` `=0.5.16`		`yes`	`yes` (2 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/ Create/ Create2
1.0	`yes`			`yes`	`yes`	`yes → AssemblyCall:Name:create2`

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. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	No

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

The modifiers and public functions are the same as the Uniswap V2 project.

The only differences for the core are:

- Explicit type changes
  - From “uint” to “uint256”
- Init code hash has been changed
  - Make sure to get the correct init code hash for your UniswapV2Pair contract.

The only differences for the periphery are:

- Explicit type changes
  - From “uint” to “uint256”

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/periphery/interfaces/IUniswapV2Migrator.sol		1	11	4	3		3	
🔍	contracts/periphery/interfaces/IWETH.sol		1	9	4	3		10	💰
🔍	contracts/periphery/interfaces/IUniswapV2Router02.sol		1	50	6	4		16	💰
🔍	contracts/periphery/interfaces/IUniswapV2Router01.sol		1	154	4	3		48	💰
🔍	contracts/periphery/interfaces/IERC20.sol		1	28	7	5		19	
📝	contracts/periphery/UniswapV2Router02.sol	1		505	326	282	14	310	💰🔷
📚	contracts/periphery/libraries/UniswapV2LiquidityMathLibrary.sol	1		147	117	83	15	54	
📚	contracts/periphery/libraries/SafeMath.sol	1		17	17	12	1	4	
📚	contracts/periphery/libraries/UniswapV2Library.sol	1		116	88	69	9	71	🧱
📚	contracts/periphery/libraries/UniswapV2OracleLibrary.sol	1		41	33	20	8	14	
🔍	contracts/core/interfaces/IUniswapV2Callee.sol		1	10	4	3		3	
🔍	contracts/core/interfaces/IUniswapV2Pair.sol		1	96	7	5		55	
🔍	contracts/core/interfaces/IUniswapV2Factory.sol		1	21	6	4		17	
🔍	contracts/core/interfaces/IUniswapV2ERC20.sol		1	44	7	5		27	
🔍	contracts/core/interfaces/IERC20.sol		1	28	7	5		19	
📝	contracts/core/UniswapV2Pair.sol	1		228	206	170	34	184	🧱
📚	contracts/core/libraries/Math.sol	1		23	23	18	2	5	
📚	contracts/core/libraries/UQ112x112.sol	1		20	20	10	6	4	
📚	contracts/core/libraries/SafeMath.sol	1		17	17	12	1	4	
📝	contracts/core/UniswapV2ERC20.sol	1		114	94	79	1	61	💻🧱🔪
📝	contracts/core/UniswapV2Factory.sol	1		51	51	41	2	53	💻🧱🌀
📝📚🔍	**Totals**	**11**	**10**	**1730**	**1048**	**836**	**93**	**981**	💻💰🔷🧱🔪🌀

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	Status
#1	UniswapV2Factory	Missing Zero Address Validation (missing-zero-check)	17, 24, 47	Check that the address is not zero	Open
#2	UniswapV2Pair	Missing Zero Address Validation (missing-zero-check)	78	Check that the address is not zero	Open
#3	UniswapV2Router02	Missing Zero Address Validation (missing-zero-check)	23	Check that the address is not zero	Open
#4	All	Old Compiler Version	—	The contracts use an outdated compiler version which is not recommended for deployment as it may be susceptible to known vulnerabilities.	Open

#5	UniswapV2LiquidityMathLibrary.sol	Overflow/ underflow	68-76, 34	Since the pragma version is not above 0.8.x the overflow/underflow will not be handled by default when the contract is using raw mathematical operations. Use instead the safemath library to make sure that there is no desired overflow/underflow. In general replace every raw mathematical operations in every files where the pragma version is below 0.8.x. Also it is recommended to use safeMath library functions (divide, etc.).	Open

Informational issues

Issue	File	Type	Line	Description	Status
#1	UniswapV2Library	Check init code hash	31	Checking the init hash code for the modified UniswapV2Pair is recommended. Also, test the init code hash that the contracts are working smoothly.	Open
#2	All	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc. Write natspec comments to all of your functions are recommended.	Open
#3	All files	SPDX License	Top of file	Add a SPDX License into the source code.	Open
#4	UniswapV2OracleLibrary.sol	Underflow?	31-38	The calculation in the “currentCumulativePrices” is intended to overflow. When the value will go into the negative area it will be an underflow. Is that als intended?	Open

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.

13. June 2023:

- 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

SolidProof_io

@solidproof_io

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

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	NOT 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