SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment
29. November, 2021

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	29. November 2021	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Fantom FTM

Website
https://raven.scarecrow.fi/

Telegram
https://t.me/scarecrowfinancebrpt
https://t.me/scarecrowfinance

Twitter
https://twitter.com/scarecrowdefi

Discord
https://discord.gg/SsxBSPQXDt

Description

The Raven is a layer 2 of ScareCrow Finance, aiming to bring high incentives and big burns to our main SCARE token, helping it to become deflationary. Our goal is to bring high rewards and incentives to holders of our native RAVEN and SCARE tokens!

Project Engagement

During the 23rd of November 2021, **Raven 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

Raven: https://ftmscan.com/address/0x175cbf2809acfd7521fdd387d65aac523fe4076f#code

MasterChef: https://ftmscan.com/address/0x2639779d6ca9091483a2a7b9a1fe77ab83b90281#code

Timelock: https://ftmscan.com/address/0xc1299d52c38c588c99df9da1fc4b715e7e7585cf#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)
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:

```
Address.sol
BEP20.sol
Context.sol
IBEP20.sol
IUniswapV2Factory.sol
IUniswapV2Pair.sol
IUniswapV2Router02.sol
```

```
ReentrancyGuard.sol
SafeBEP20.sol
SafeMath.sol
```

Tested Contract Files

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

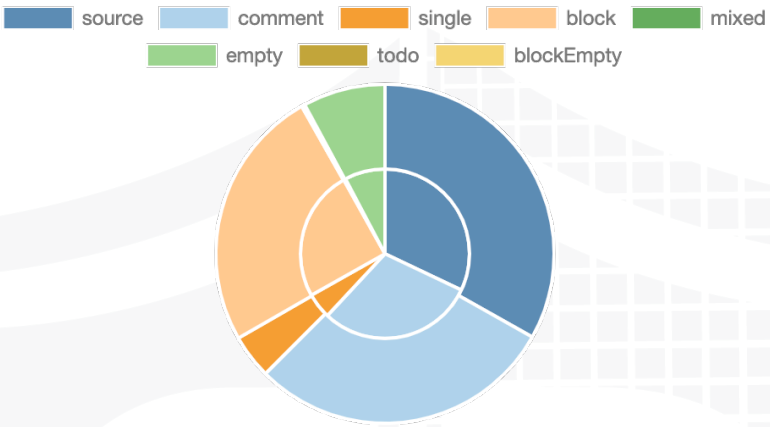
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

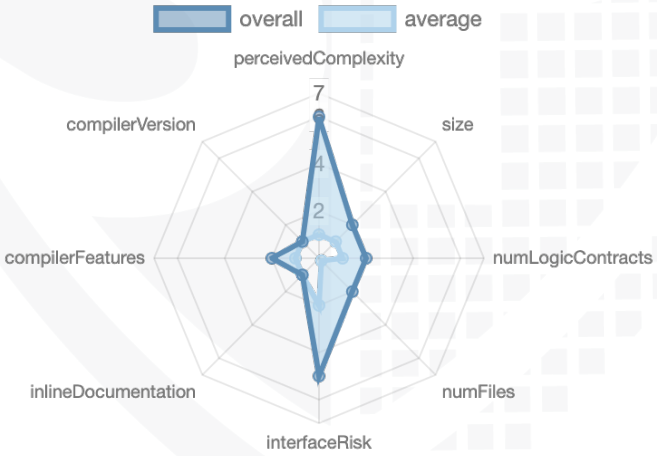
File Name	SHA-1 Hash
contracts/IBEP20.sol	7a33f5e0ff19f7ec6489687c2b1ec3519b775321
contracts/BEP20.sol	149358ecd07327c44047745d5e0702d396ed9500
contracts/IUniswapV2Pair.sol	c1ca96bb12261bdfebec552a6ba59084b30b44b3
contracts/Context.sol	a34cc2179b2da819d60afa9d711d0094d5a72799
contracts/MasterChef.sol	e5b4cad2c75c8be47107abab029e0cb9a7906aea
contracts/Raven.sol	3571a96481af5f05686e2de1e016bed4af163731
contracts/SafeBEP20.sol	fd04404c0141a9d5807076b77bc7cb9c9b4a2422
contracts/IUniswapV2Factory.sol	7a04a56d81a11303fe2b01f24ad5ea8251eaa479
contracts/Timelock.sol	8fef29a499a33dbc1ec35bbe6d6b043fa18fa9ff
contracts/Address.sol	c03046a3df309f7bab7fd49bafaa2d755c20e1d1
contracts/SafeMath.sol	252b3caeb72fa4bde1cf723d04677a593bd82d36
contracts/Ownable.sol	171ca6e81058798295a90dc7f98fea18370d52a1
contracts/IUniswapV2Router02.sol	189cf2bf5b433caf92a4642c22b5df8b7de153bf
contracts/ReentrancyGuard.sol	219a1c1c0febcd3a0b81af2304558232b87b6227

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	4	4	5	3

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	133	7

Version	External	Internal	Private	Pure	View
1.0	92	161	6	39	50

State Variables

Version	Total	Public
1.0	52	37

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts

1.0	`>=0.6.4`
	`>=0.4.0`
	`>=0.5.0`
	`>=0.6.0`
	`<0.8.0`
	`0.6.12`
	`>=0.6.2`
	`<0.8.0`
	`>=0.6.2`
			`yes`	`yes`
				(3 asm blocks)	

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

Scope of Work

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. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Inheritance Graph

v1.0

Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

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

Raven

1. approve
2. decreaseAllowance
3. delegate
4. delegateBySig
5. increaseAllowance
6. mint
7. mint
8. renounceOwnership
9. setExcludedFromAntiWhale
10. setExcludedFromTransferTax
11. transfer
12. transferFrom
13. transferOperator
14. transferOwnership
15. updateMaxTransferAmountRate
16. updateMinAmountToLiquify
17. updateScarecrowFinanceRouter
18. updateSwapAndLiquifyEnabled
19. updateTransferTaxRate

Masterchef

1. add
2. deposit
3. emergencyWithdraw
4. massUpdatePools
5. renounceOwnership
6. set
7. setBuyBackAddress
8. setDevAddress
9. setFeeAddress1
10. setFeeAddress2
11. setFeeAddress3
12. transferOwnership
13. updateEmissionRate
14. updateMaxSupply
15. updatePool
16. updateStartBlock
17. withdraw

Timelock

1. acceptAdmin
2. cancelTransaction
3. executeTransaction
4. queueTransaction
5. setDelay
6. setPendingAdmin

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified	File
Deployer cannot mint	✓	✓	✓	Main
Comment	Line: -			

Max / Total Supply:

Comments:

v1.0

- onlyOwner can mint
  - Owner of Raven token is MasterChef

Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot lock	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Comments:

v1.0

- Amount will transfer to burn address in swapAndLiquify function
  - triggerScareBuyback
- Deployer can set maxTransferAmountRate between 30-10000
- transferTaxRate can be so between 0 - 1000 (10%)
- Line 138
  - Unnecessary line of code

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	-	-	-

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

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

Modifiers

Raven

- onlyOwner
  - mint
- onlyOperator

```
updateTransferTaxRate
updateMaxTransferAmountRate
updateMinAmountToLiquify
setExcludedFromAntiWhale
setExcludedFromTransferTax
updateSwapAndLiquifyEnabled
updateScarecrowFinanceRouter
transferOperator
```

MasterChef

- onlyOwner

```
add
set
```

```
updateEmissionRate
updateStartBlock
updateMaxSupply
```

  -

Comments

- updateMinAmountToLiquify
  - Operator can set minAmountToLiquify without any limitations

CallGraph

Source Units in Scope
v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/IBEP20.sol	————	1	94	23	17	66	21	————
📝	contracts/BEP20.sol	1	————	298	298	98	170	91	————
🔍	contracts/IUniswapV2Pair.sol	————	1	54	9	5	1	55	————
🎨	contracts/Context.sol	1	————	24	24	10	12	1	🔆
📝	contracts/MasterChef.sol	1	————	346	346	269	47	220	💠
📝	contracts/Raven.sol	1	————	580	550	327	155	247	💻💰🧮📮
📚	contracts/SafeBEP20.sol	1	————	75	74	33	32	25	————
🔍	contracts/IUniswapV2Factory.sol	————	1	19	8	4	1	17	————
📝📚	contracts/Timelock.sol	2	————	354	354	145	159	91	💰🧮🔆
📚	contracts/Address.sol	1	————	189	169	78	113	47	💻👥
📚	contracts/SafeMath.sol	1	————	214	214	61	139	16	🔆
🎨	contracts/Ownable.sol	1	————	68	68	27	33	24	————
🔍	contracts/IUniswapV2Router02.sol	————	2	142	7	4	1	64	💰
🎨	contracts/ReentrancyGuard.sol	1	————	62	62	15	38	5	————
📝📚🔍🎨	**Totals**	**11**	**5**	**2519**	**2206**	**1093**	**967**	**924**	💻💰💠👥🧮📮🔆

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 found -

High issues

- no high issues found -

Medium issues

- no medium issues found -

Low issues

Issue	File	Type	Line	Description
#1	All	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	Timelock	A floating pragma is set	8	The current pragma Solidity directive is „">=0.6.0 < 0.8.0"".
#3	MasterChef	Missing Zero Address Validation (missing-zero-check)	91, 87, 88, 89, 90, 286,	Check that the address is not zero
#4	Timelock	Missing Zero Address Validation (missing-zero-check)	261, 323, 290	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	Raven	Don't use raw math arithmetic	134	Use Safemath library instead

#2	MasterChef	Conformity to Solidity naming conventions	80, 81	Follow the Solidity [naming convention](https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions)

Audit Comments

29. November 2021:

- Read report 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

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

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	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