



SOLIDProof
Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Rabbits.Money

Audit

Security Assessment
26. April, 2023

For



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
Commented Code exist	28
Audit Comments	28
SWC Attacks	29

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 Uniswap, 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	24. April 2023	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary

Network

Arbitrum

Website

<https://www.rabbits.money/>

Twitter

<https://twitter.com/RabbitsdotMoney>



Description

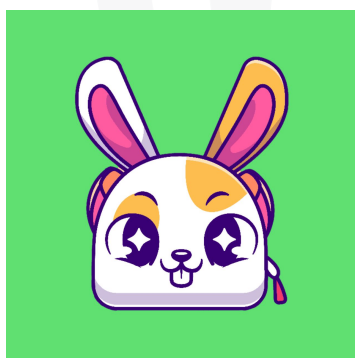
Rabbits.Money is a new and fun journey into the world of DeFi. Inspired by past successful node projects such as Dairy.Money, we came up with a new twist to tokenomics so that the project is more sustainable, while also keeping the simplicity that was meant by these original node projects.

We are very glad to introduce you to Rabbits.Money and our Rabbit nodes!

Project Engagement

During the Date of 24 April 2023, **Rabbits Money 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

- <https://arbiscan.io/address/0xf3a4354d623246704f9438dd3043c5a247ae592e#code>

Note - This Audit report consists of security analysis of the Rabbits.Money smart contracts, functional testing (or unit testing) of the contract's logic was not included in this analysis.

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 as possible.
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/access/Ownable.sol	2
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	3
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	2
@openzeppelin/contracts/utils/math/SafeMath.sol	2

Tested Contract Files

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

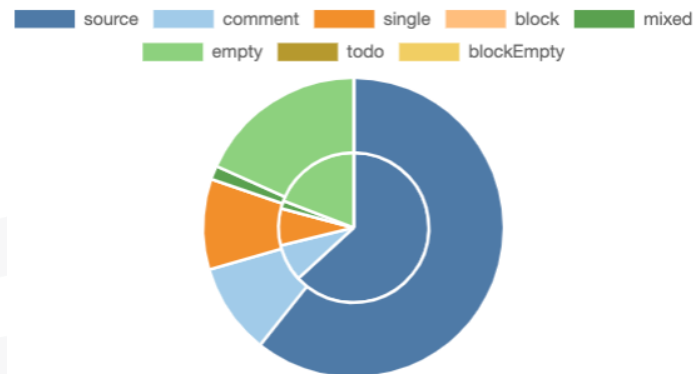
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

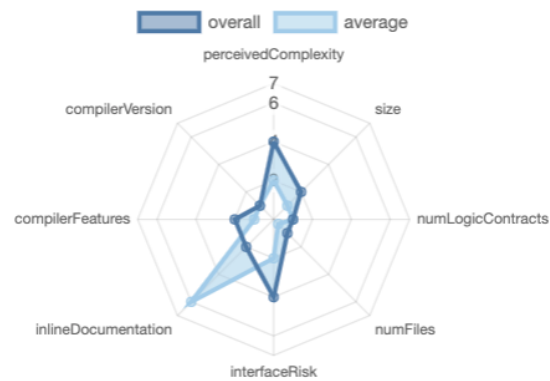
File Name	SHA-1 Hash
contracts/ CarrotToken.sol	aa97585181887e8212b1ab603c5ae23df633 5aa6
contracts/ITEST.sol	199958a89c1e14a4dfdf8077f20880363f81f6f 1
contracts/ CarrotFarm.sol	a118ca719f34d703bef64de5e099743f794a96 b4

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components












 Public	 Payable
33	3

External	Internal	Private	Pure	View
19	30	2	1	9

StateVariables

Total	 Public
22	12

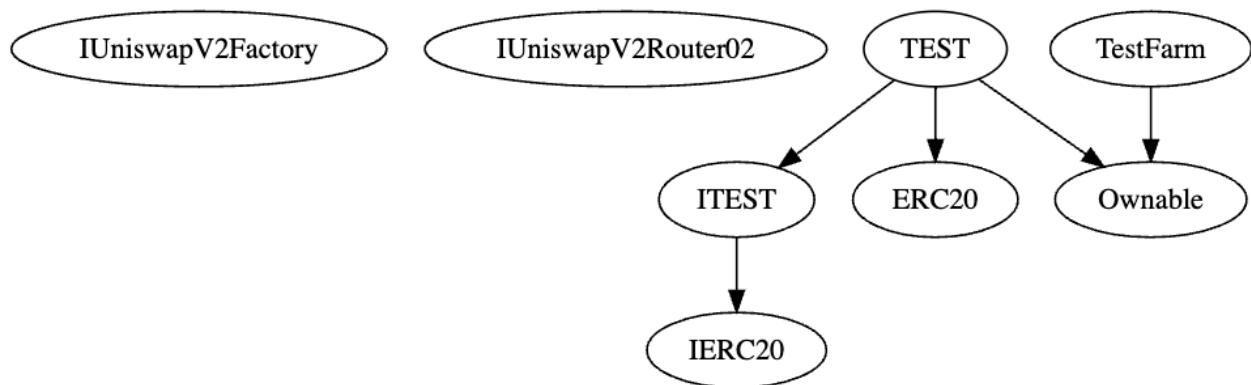
Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
0.8.15		yes			
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTrecover	 New/Create/Create2
 TryCatch	Σ Unchecked				



Inheritance Graph

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



Is contract an upgradeable

Name	
Is contract an upgradeable?	No



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

1. awardNode (0x75166473)

2. bondNode (0xaa11aecd)

3. buyNode (0xedd47fe6)

4. calculateTax (0xb9b2735c)

5. claim (0x4e71d92d)

6. compoundNode (0x29a5c72b)

7. renounceOwnership (0x715018a6)

8. setBondDiscount (0x357c8acd)

9. setBondNodeStartTime (0xb3bc10cd)

10. setBondTax (0x4260aabc)

11. setDailyInterest (0xbec82034)

12. setPlatformState (0xe3b57275)

13. setTestAddr (0xa70d2919)

14. setTestTax (0xc74631cf)

15. setTreasuryAddr (0xa7e05b9c)

16. transferOwnership (0xf2fde38b)

Deployer/Authority cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✗
Max / Total Supply	N/A		

Comments:

v1.0

- Max supply will be decided at the time of final deployment, and the owner can add controller addresses, and these addresses will be able to mint unlimited tokens

Deployer/Authority cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer can lock	✓	✓	✗
Deployer can burn	✓	✓	✗

Comments:

v1.0

- Controller addresses can lock user funds by pausing the trading in the contract
- The owner can add controller addresses, and these addresses will be able to burn tokens from any address/wallet that holds them which is not recommended. It is a high centralisation risk

Deployer/Authority cannot pause the contract

Name	Exist	Tested	Status
Deployer can pause	✓	✓	✗

Comments:

v1.0

- Controller addresses can pause contract



Deployer/Authority 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

- The tax in the farm contract can be set without any limitations, and if it is set to 100% or above then the users will not get any reward. But because the tax is automatically burned and not credited to any wallet, the owner has nothing to gain by setting the tax to a 100%

Deployer/Authority 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

CarrotFarm



◆	setBondDiscount
Ⓜ	onlyOwner
◆	setTreasuryAddr
Ⓜ	onlyOwner
◆	setTestAddr
Ⓜ	onlyOwner
◆	setTestTax
Ⓜ	onlyOwner
◆	setBondTax
Ⓜ	onlyOwner
◆	setPlatformState
Ⓜ	onlyOwner
◆	setDailyInterest
Ⓜ	onlyOwner
◆	setBondNodeStartTime
Ⓜ	onlyOwner
◆	buyNode 💰
◆	bondNode 💰
◆	awardNode
Ⓜ	onlyOwner
◆	compoundNode
◆	calculateTax
◆	claim 💰

Ownership Privileges

- Set/Change test token address, which may result in users receiving a different token while claiming the yield
- Owner can stop users from buying node by setting the live status of the contract to false
- Award nodes to any user with a maximum of 100 nodes
- Set daily interest to any arbitrary value

- Set/Change bond node start time to any arbitrary value. So, if the owner wants then users may be able to bond nodes right after buying nodes or they may never be able to bond nodes.
- 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

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



Source Units in Scope

v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/CarrotToken.sol	1	2	150	102	58	28	55
contracts/ITEST.sol	1	1	10	6	4	1	7
contracts/CarrotFarm.sol	1	1	313	313	238	24	195
Totals	2	3	473	421	300	53	257

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	testf.sol	Missing Zero Address Validation (missing-zero-check)	122, 125, 201	Check that the address is not zero
#2	testf.sol	State variable visibility is not set	44	It is best practice to set the visibility of state variables explicitly
#3	testf.sol	Missing Events Arithmetic	115-155	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	testf.sol	State variables that could be declared immutable (immutable-states)	28, 30, 35, 36	Add the `immutable` attributes to state variables that never change
#2	testf.sol	Unused state variables	32	Remove unused state variables
#3	testf.sol	Functions that are not used	146	Remove unused functions. Before removing check the function, it could be possible, that you forget to implement it into the contract

#4	All	NatSpec documentation missing	—	If you started to comment your code, also comment all other functions, variables etc.
----	-----	-------------------------------	---	---

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

File	Line	Comment
test.sol	23-37	<pre>// function addLiquidityETH(// address token, // uint256 amountTokenDesired, // uint256 amountTokenMin, // uint256 amountETHMin, // address to, // uint256 deadline //) // external // payable // returns (// uint256 amountToken, // uint256 amountETH, // uint256 liquidity //);</pre>

Recommendation

Remove the commented code, or address them properly.

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.

26. April 2023:

- There is still an owner (Owner still has not renounced ownership)
- 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
- Unit tests with 100% code coverage was not provided to **SolidProof** so we cannot ensure complete functional correctness of the code's logic.
- We recommend **Rabbits.Money** team to conduct unit and fuzz tests thoroughly to rule out possibilities of an unwanted logical and calculation errors.

SWC Attacks

ID	Title	Relationships	Status
SW C-1 36	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-1 35	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-1 34	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-1 33	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-1 32	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-1 31	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-1 30	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-1 29	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-1 28	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-1 27	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-1 25	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-1 24	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-1 23	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-1 22	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-1 21	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-1 20	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SW C-11 9	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-11 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SW C-11 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 1	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-1 09	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-1 08	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	NOT PASSED
SW C-1 07	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-1 06	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-1 05	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-1 04	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-1 03	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SW C-1 02	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-1 01	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SW C-1 00	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

*Solid
Proofed*

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**


MADE IN GERMANY