SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

12. January, 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	12. January 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://sumari.finance/

Twitter
https://twitter.com/SumariFinance

Github
https://github.com/SumariFinance/

Reddit
https://www.reddit.com/r/SumariFinance/

Medium
https://sumarifinance.medium.com/

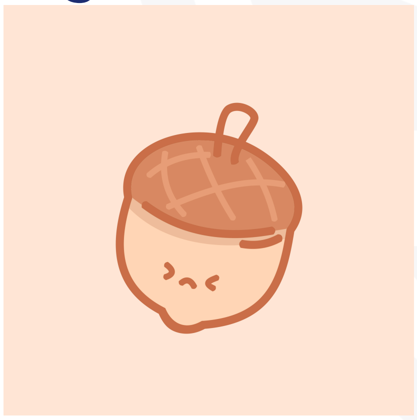
Description

Sumari Finance is a yield optimizer platform on several blockchains, offering users the opportunity to increase their crypto portfolio through the marvel of compounding. On Sumari Finance, you can maximize your ROI thanks to unrivaled APYs that are tamperproof and safe. Sumari Finance uses multiple strategies executed through smart contracts to ensure users get the maximum rewards possible from their preferred yield farming protocols, liquidity pools, and more. To avail themselves of these opportunities, users need to deposit their assets in the Sumari Vaults, a financial product that's linked to the yield optimizing services on the protocol. The deposited tokens gradually increase as interests from the selected yield farming option are compounded. Tokens deposited in the Sumari Vaults can be taken out by the user at any time – zero locking mechanisms enforced.

Project Engagement

During the 11th of January 2022, **Sumari 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://bscscan.com/address/0x6eea539b9397bdb673eacbd84ba501dbbf95e016#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.

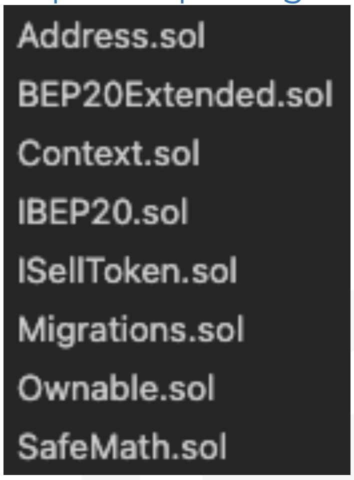
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program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



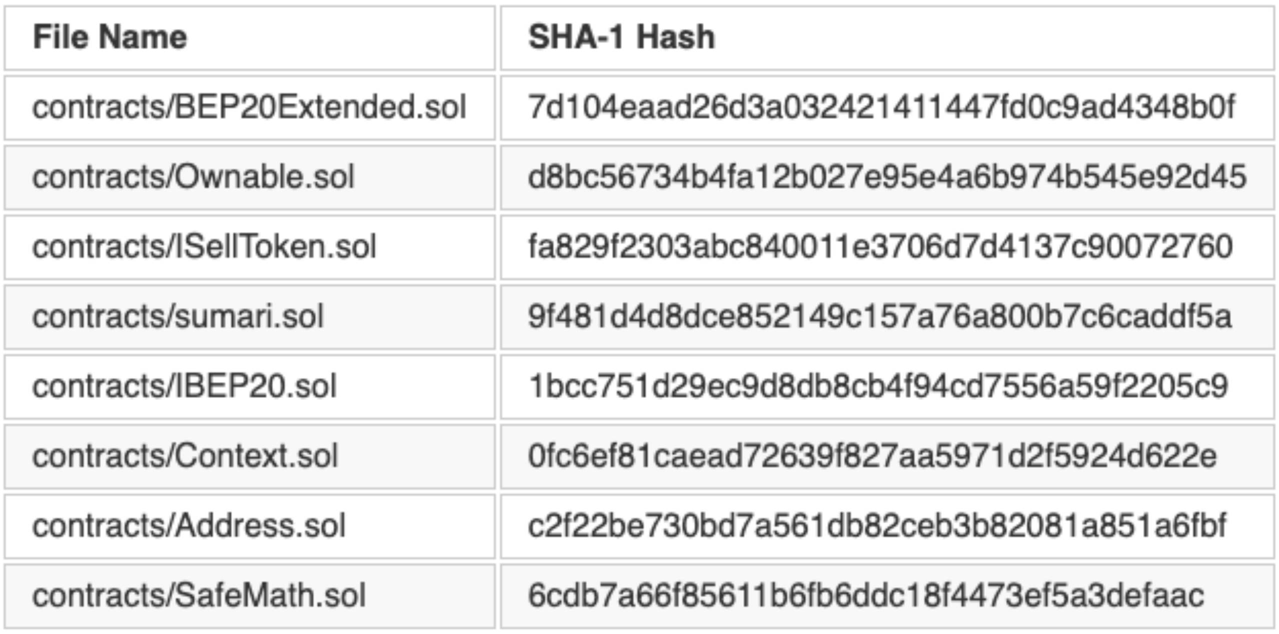
Address.sol
BEP20Extended.sol
Context.sol
IBEP20.sol
ISellToken.sol
Migrations.sol
Ownable.sol
SafeMath.sol

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

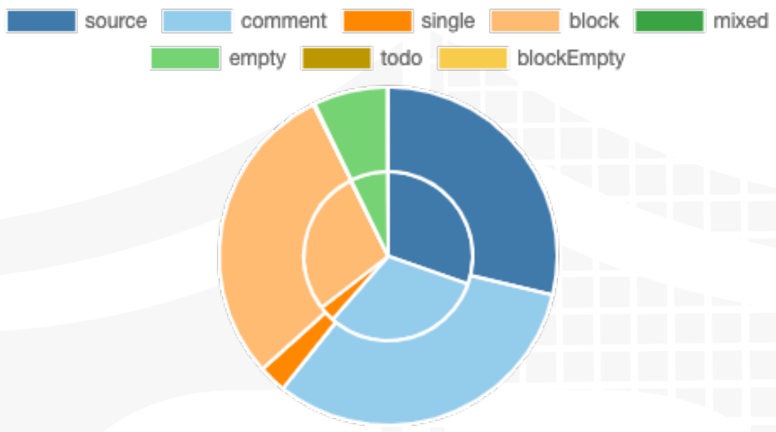


File Name	SHA-1 Hash
contracts/BEP20Extended.sol	7d104eaad26d3a032421411447fd0c9ad4348b0f
contracts/Ownable.sol	d8bc56734b4fa12b027e95e4a6b974b545e92d45
contracts/ISellToken.sol	fa829f2303abc840011e3706d7d4137c90072760
contracts/sumari.sol	9f481d4d8dce852149c157a76a800b7c6caddf5a
contracts/IBEP20.sol	1bcc751d29ec9d8db8cb4f94cd7556a59f2205c9
contracts/Context.sol	0fc6ef81caead72639f827aa5971d2f5924d622e
contracts/Address.sol	c2f22be730bd7a561db82ceb3b82081a851a6fbf
contracts/SafeMath.sol	6cdb7a66f85611b6fb6ddc18f4473ef5a3defaac

Metrics

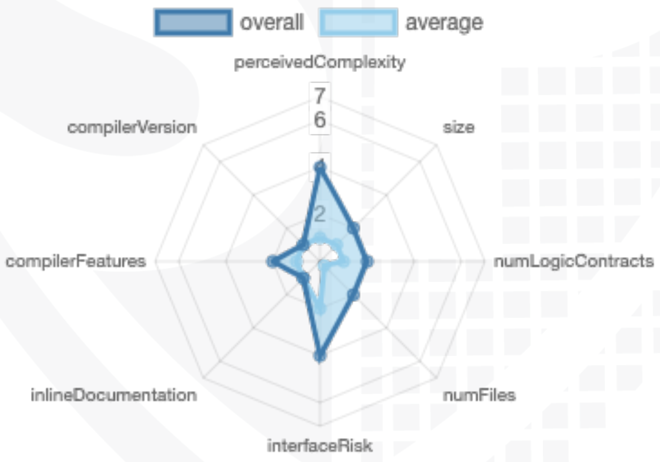
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	4	2	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	46	0

Version	External	Internal	Private	Pure	View
1.0	18	77	1	13	27

State Variables

Version	Total	Public
1.0	22	10

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`>=0.4.0` `0.6.12` `>=0.6.2` `<0.8.0`			`yes` (3 asm blocks)	

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

Scope of Work

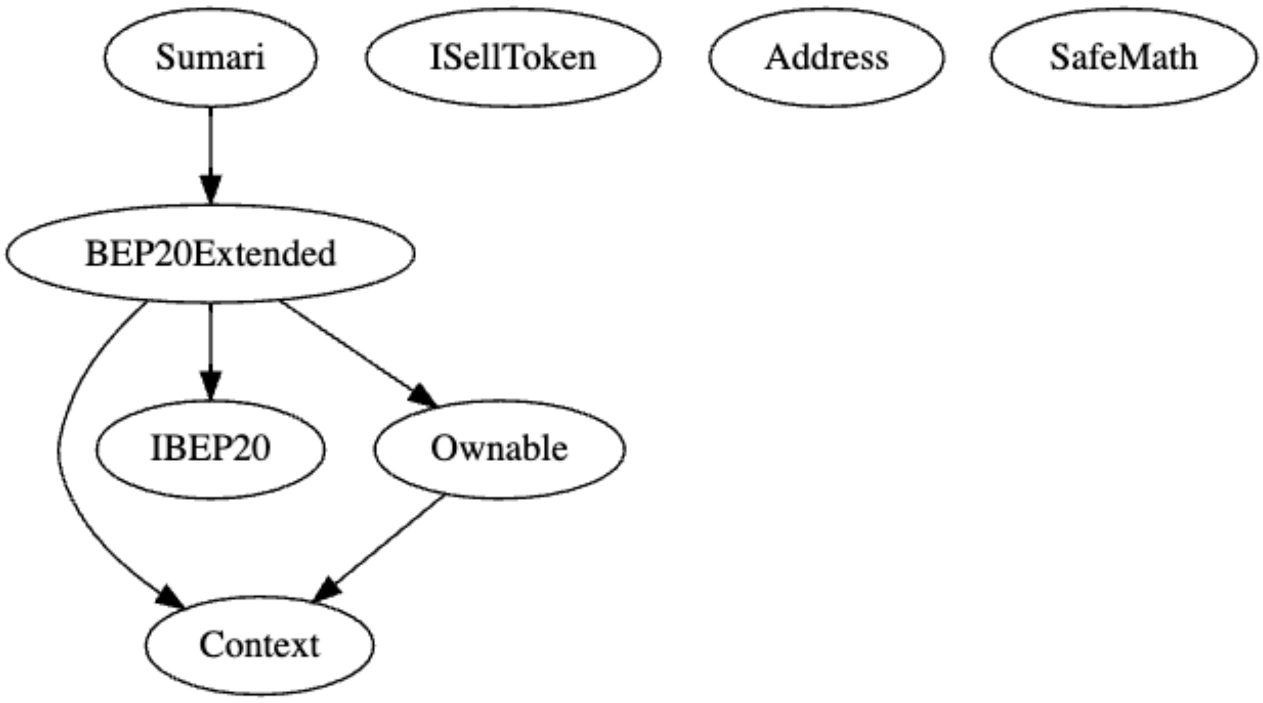
The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

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



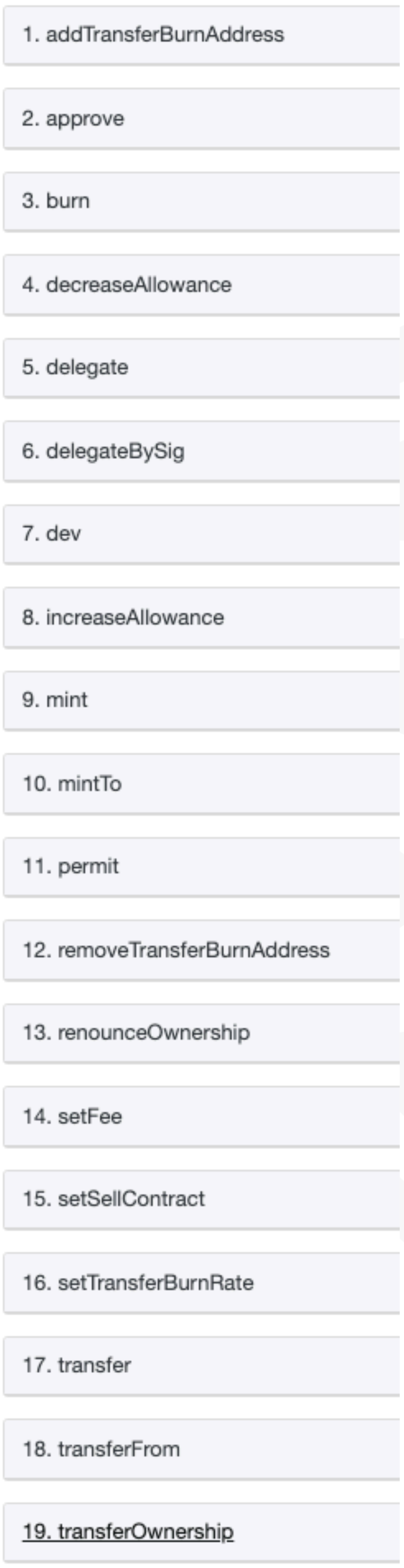
Verify Claims

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



1. addTransferBurnAddress
2. approve
3. burn
4. decreaseAllowance
5. delegate
6. delegateBySig
7. dev
8. increaseAllowance
9. mint
10. mintTo
11. permit
12. removeTransferBurnAddress
13. renounceOwnership
14. setFee
15. setSellContract
16. setTransferBurnRate
17. transfer
18. transferFrom
19. transferOwnership

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified
Deployer cannot mint	✓	✓	✗

Max / Total Supply: 200.000.000

Comments:

v1.0

- Deployer can mint new tokens as long as total supply + minting amount is lower equal to cap

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Everybody can burn tokens
- Deployer can lock user funds by setting transferBurnRate to 0 if recipient is burn address or fee is enabled

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

- onlyOwner
  - mintTo
  - Mint
  - setSellContract
- Only dev address
  - dev
    - Set dev address
  - setFee
    - Enable fees
  - setTransferBurnRate
  - addTransferBurnAddress
  - removeTransferBurnAddress

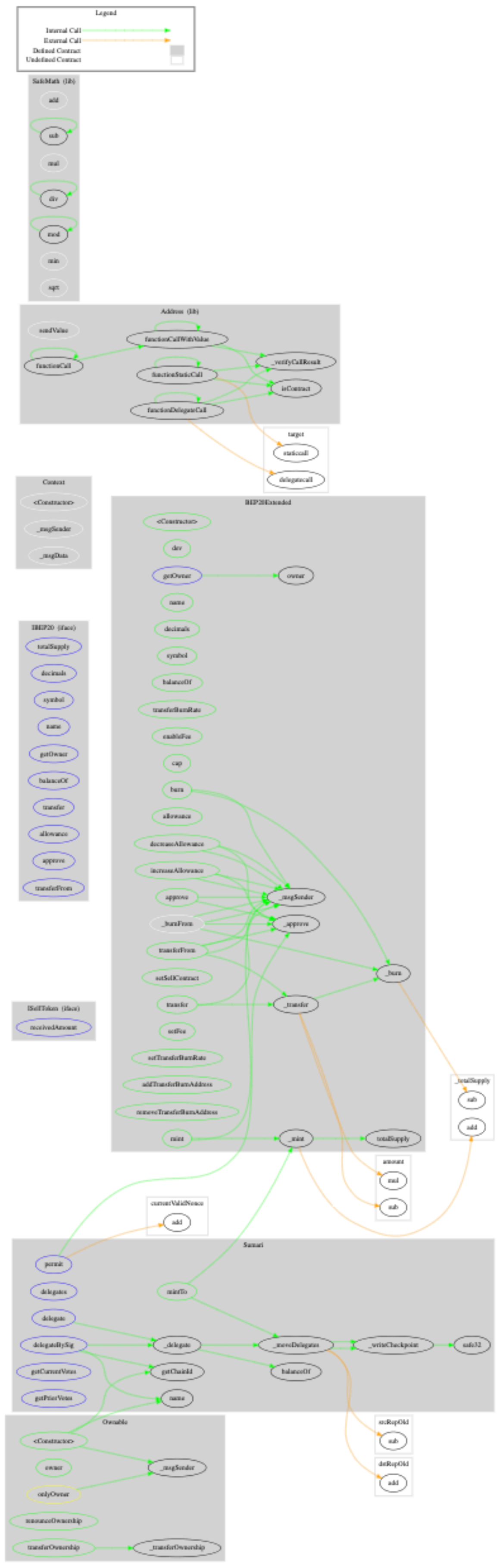
Comments

- Developer can set following state variables without any limitations
  - _transferBurnRate

- Deployer can enable/disable following state variables
  - _transferBurnAddresses

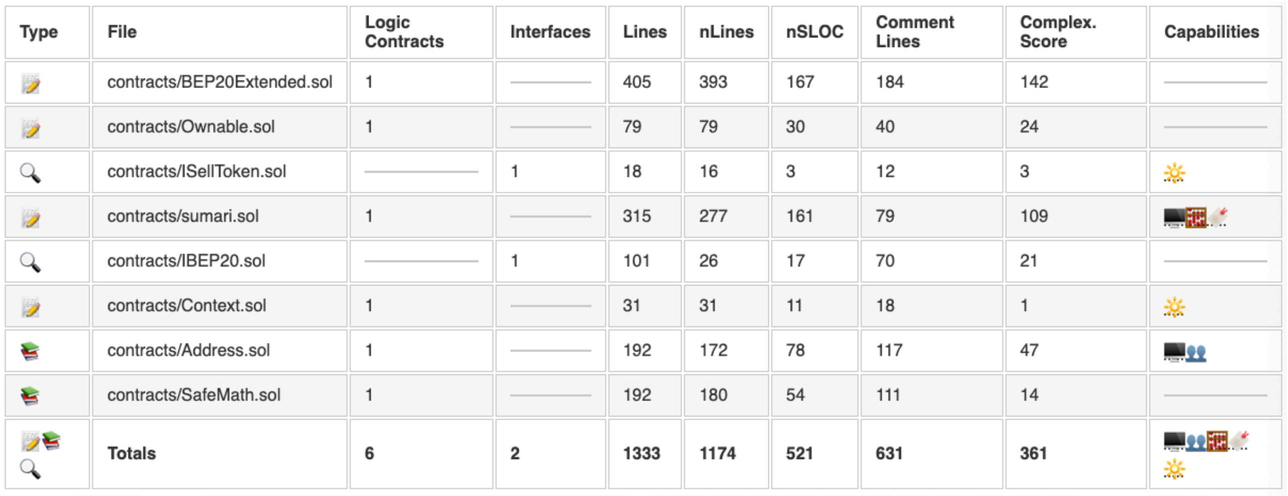
- If the contract owner has hired a developer to deploy the contract, the contract owner cannot set a new devaddr. Only devaddr can set a new devaddr (see above)

CallGraph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
mod
min
sqrt
Address (lib)
sendValue
functionCallWithValue
functionCall
_verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
Context
<Constructor>
_msgSender
_msgData
BEP20Extended
<Constructor>
dev
getOwner
owner
name
decimals
symbol
balanceOf
transferBurnRate
enableFee
cap
burn
allowance
decreaseAllowance
increaseAllowance
approve
_msgSender
_burnFrom
_approve
transferFrom
_burn
setSellContract
transfer
_transfer
setFee
setTransferBurnRate
addTransferBurnAddress
removeTransferBurnAddress
mint
_mint
totalSupply
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
ISellToken (iface)
receivedAmount
_totalSupply
sub
add
amount
mul
sub
currentValidNonce
add
Samari
permit
delegates
mintTo
delegate
delegateBySig
_delegate
_moveDelegates
_writeCheckpoint
safe32
getCurrentVotes
getChainId
balanceOf
getPriorVotes
name
Ownable
<Constructor>
owner
_msgSender
onlyOwner
renounceOwnership
transferOwnership
_transferOwnership
srcRepOld
sub
dstRepOld
add

Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/BEP20Extended.sol	1	—	405	393	167	184	142	—
📝	contracts/Ownable.sol	1	—	79	79	30	40	24	—
🔍	contracts/ISellToken.sol	—	1	18	16	3	12	3	☀
📝	contracts/sumari.sol	1	—	315	277	161	79	109	🖥🧮💉
🔍	contracts/IBEP20.sol	—	1	101	26	17	70	21	—
📝	contracts/Context.sol	1	—	31	31	11	18	1	☀
📚	contracts/Address.sol	1	—	192	172	78	117	47	🖥👥
📚	contracts/SafeMath.sol	1	—	192	180	54	111	14	—
📝📚🔍	**Totals**	**6**	**2**	**1333**	**1174**	**521**	**631**	**361**	🖥👥🧮💉☀

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

Issue	File	Type	Line	Description
#1	Main	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	BEP20Extended	A floating pragma is set	7	The current pragma Solidity directive is „">=0.4.0"".
#3	Address	A floating pragma is set	6	The current pragma Solidity directive is „">=0.6.2 <0.8.0 "".
#4	Context, IBEP20, Ownable, SafeMath	A floating pragma is set	6	The current pragma Solidity directive is „">=0.4.0"".

#5	BEP20Extended	Missing Zero Address Validation (missing-zero-check)	81, 260	Check that the address is not zero
#6	BEP20Extended	Local variables shadowing	379, 165, 71	Rename the local variables that shadow another component
#7	Sumari	Local variables shadowing	98	Rename the local variables that shadow another component
#8	BEP20Extended	Tautology or contradiction	319	Fix the incorrect comparison by changing the value type or the comparison

Informational issues

Issue	File	Type	Line	Description
#1	BEP20Extended	Functions that are not used	396	Remove unused functions
#2	BEP20Extended	Unreachable code	319-321	If-condition is not reachable because _balances holds only uint256 so it cannot be a negative number
#3	BEP20Extended	Misspelling	318	receivedAmount is misspelled Write receivedAmount instead of receiedAmount The v is missing
#4	Sumari	Masterchef was not provided to Solidproof	44	Remove comment in line 44 Current owner is following address 0xFe929EB2FFeFdf9CB06a25843A653eDD619f14C5

#5	BEP20Extended	Change _enableFee variable name	58	We recommend you to change the variable _enableFee to the variable _enableBurn, since you multiply the amount by _transferBurnRate to calculate the "fees" you want to burn Make sure to change it everywhere if you want to change the variable name

Commented Code exist

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

File	Line	Comment
SafeMath	133	// assert(a == b * c + a % b); // There is no case in which this doesn't hold

Recommendation

Remove the commented code, or address them properly.

Audit Comments

12. January 2022:

- SellToken contract was not provided to Solidproof
    - Please do your own research
- Masterchef was not provided to Solidproof
- Read whole 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	NOT 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**

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY