SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment
28. July, 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.

Network
Binance Smart Chain (BEP20)

Website
https://app.coffeemaker.finance/

Telegram
https://t.me/coffeeswap

Twitter
https://twitter.com/coffeeswap_DeFi

Description

Their mission is to ensure that their investors generate the maximum possible returns with our platform through the Automated APY selector pool and the limits orders to make Stake and Unstake.

Project Engagement

During the 22nd of July 2021, **CoffeeMaker 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. **CoffeeMaker Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

https://bscscan.com/address/0xF1D8E631eA3Ec269fa4D68513A50552bdE5DeCd3#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.

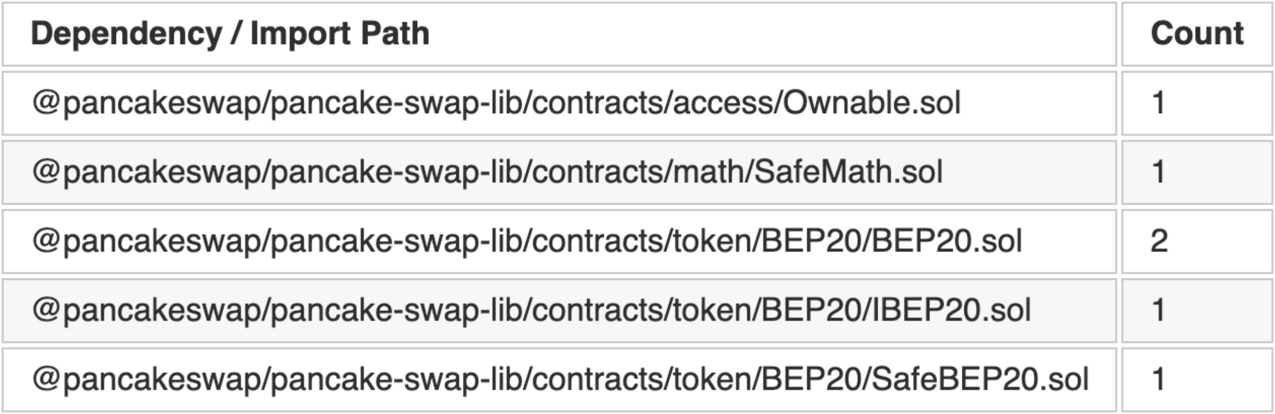
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

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:

- OpenZeppelin
    - Address
    - Ownable
    - SafeMatch

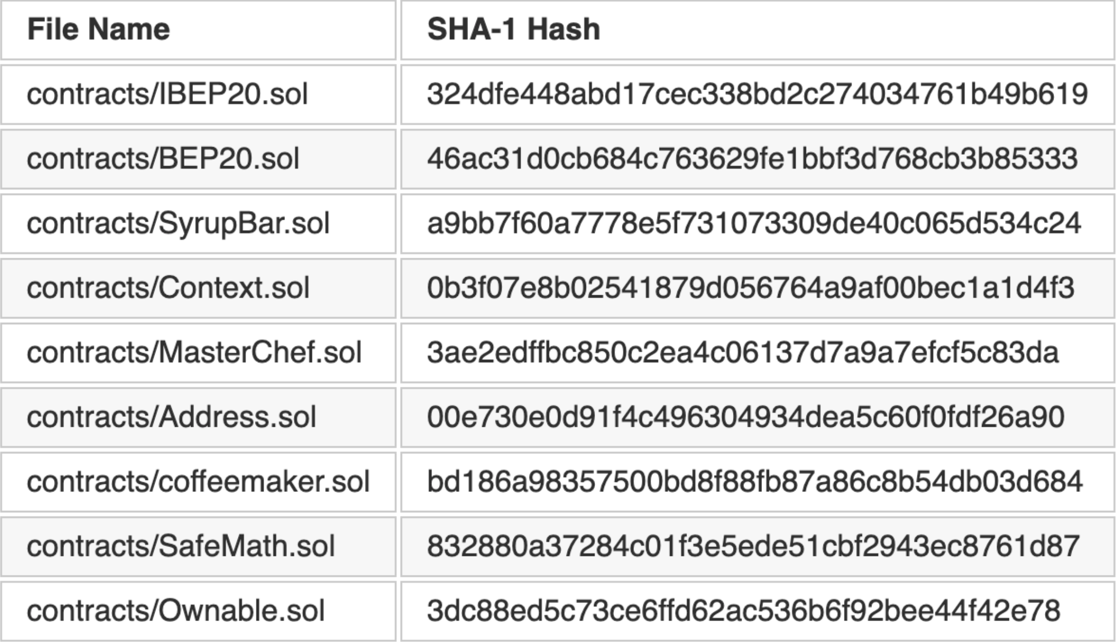


Dependency / Import Path	Count
@pancakeswap/pancake-swap-lib/contracts/access/Ownable.sol	1
@pancakeswap/pancake-swap-lib/contracts/math/SafeMath.sol	1
@pancakeswap/pancake-swap-lib/contracts/token/BEP20/BEP20.sol	2
@pancakeswap/pancake-swap-lib/contracts/token/BEP20/IBEP20.sol	1
@pancakeswap/pancake-swap-lib/contracts/token/BEP20/SafeBEP20.sol	1

Tested Contract Files

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

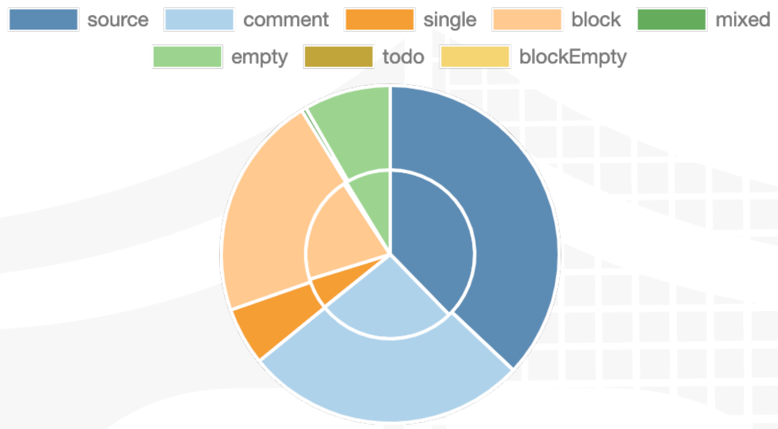
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.



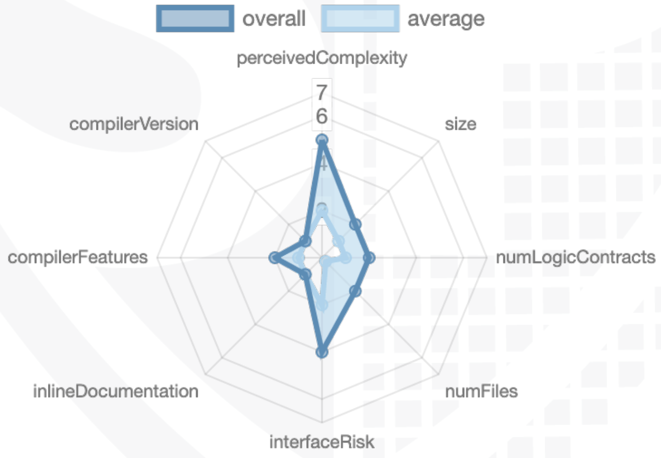
File Name	SHA-1 Hash
contracts/IBEP20.sol	324dfe448abd17cec338bd2c274034761b49b619
contracts/BEP20.sol	46ac31d0cb684c763629fe1bbf3d768cb3b85333
contracts/SyrupBar.sol	a9bb7f60a7778e5f731073309de40c065d534c24
contracts/Context.sol	0b3f07e8b02541879d056764a9af00bec1a1d4f3
contracts/MasterChef.sol	3ae2edffbc850c2ea4c06137d7a9a7efcf5c83da
contracts/Address.sol	00e730e0d91f4c496304934dea5c60f0fdf26a90
contracts/coffeemaker.sol	bd186a98357500bd8f88fb87a86c8b54db03d684
contracts/SafeMath.sol	832880a37284c01f3e5ede51cbf2943ec8761d87
contracts/Ownable.sol	3dc88ed5c73ce6ffd62ac536b6f92bee44f42e78

Metrics

Source Lines



Risk Level



Capabilities

Components

Contracts	Libraries	Interfaces	Abstract
6	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.

Public	Payable
60	0

External	Internal	Private	Pure	View
24	96	1	14	27

State Variables

Total	Public
30	21

Capabilities

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

`>=0.4.0`
`0.6.12`
`^0.6.2`
			`yes`
			(4 asm blocks)	

Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/ Create/ Create2
`yes`			`yes`	`yes`	

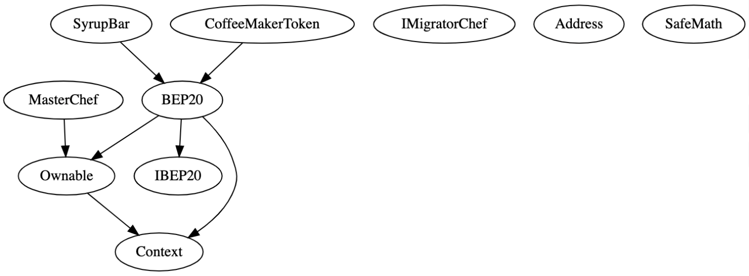
Scope of Work

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



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	✓	✓	✓

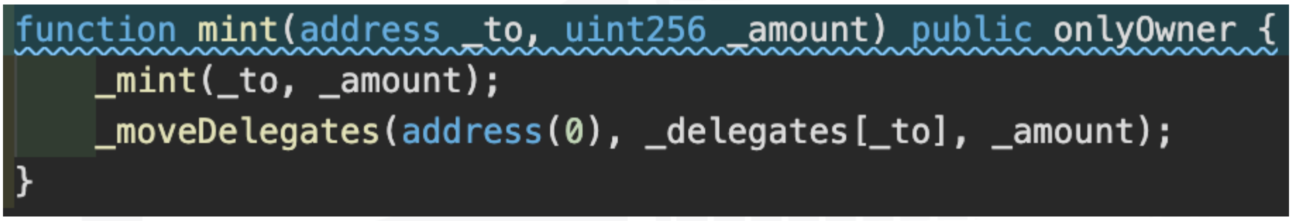
Optional implementations

Function	Description	Exist	Tested	Verified
renounceOwnership	Owner renounce ownership for more trust	✓	✓	✗

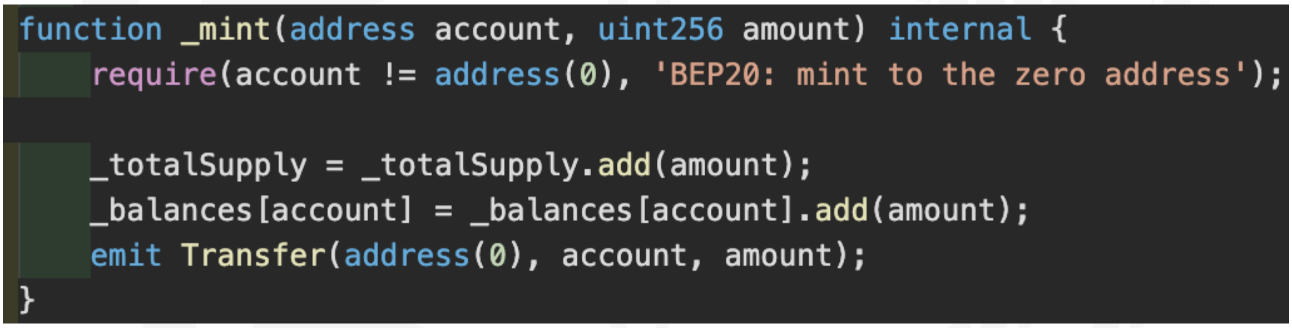
Deployer cannot mint any new tokens

Tested	Verified	File	Comment
✓	✗	Coffee maker	Line: 8 - 11

Max / Total Supply: According to contract (Address: https://bscscan.com/address/0xff484b38365896a10e68594a8754a3878af75e5b#readContract, 24. July 2021) 100.000.000.000.000.000.000.000.000 with 18 decimals



```
function mint(address _to, uint256 _amount) public onlyOwner {
    _mint(_to, _amount);
    _moveDelegates(address(0), _delegates[_to], _amount);
}
```



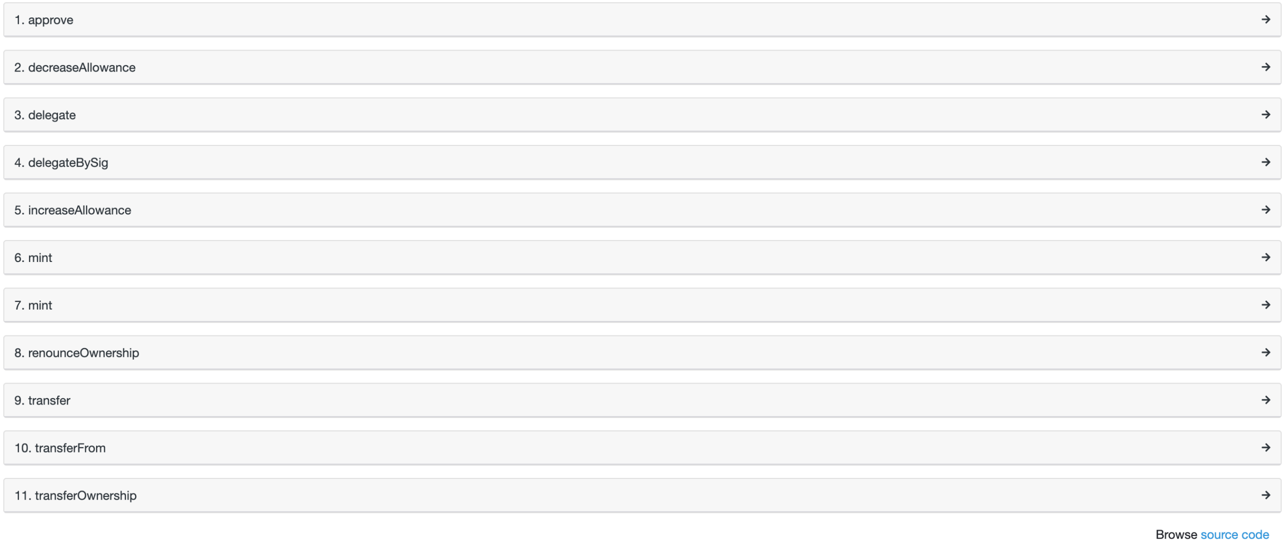
```
function _mint(address account, uint256 amount) internal {
    require(account != address(0), 'BEP20: mint to the zero address');

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}
```

Deployer cannot burn or lock user funds

Tested	Verified	No Lock function
✓	✓	✓

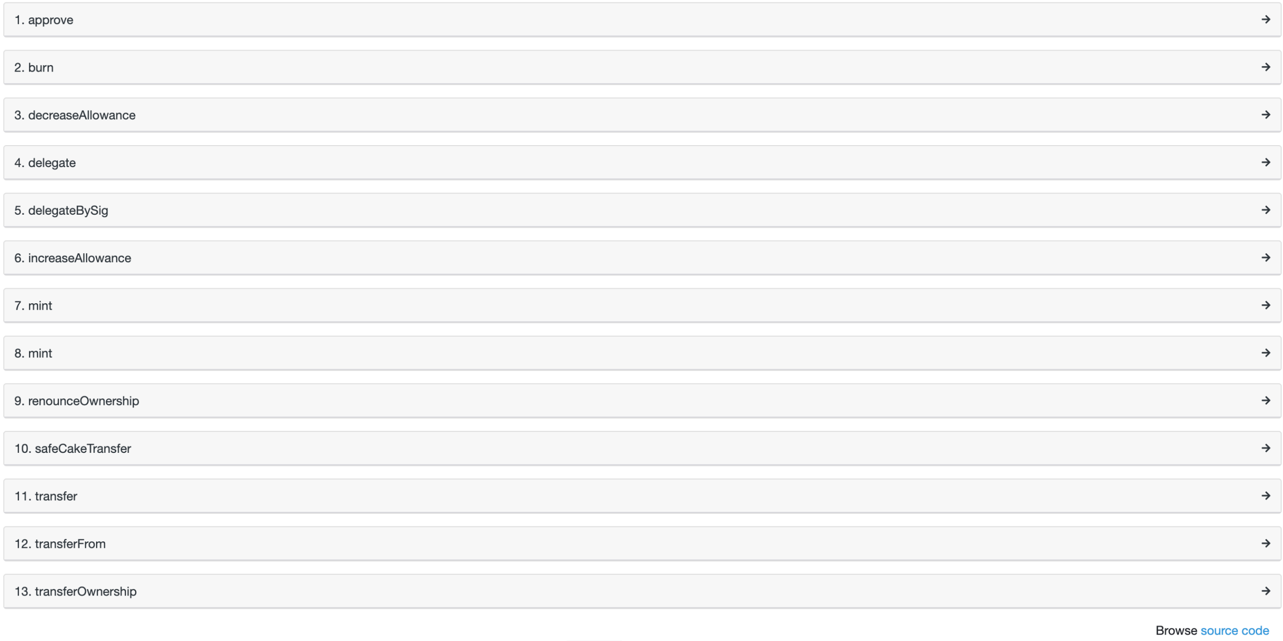
CoffeeMaker.sol



1. approve →
2. decreaseAllowance →
3. delegate →
4. delegateBySig →
5. increaseAllowance →
6. mint →
7. mint →
8. renounceOwnership →
9. transfer →
10. transferFrom →
11. transferOwnership →

Browse source code

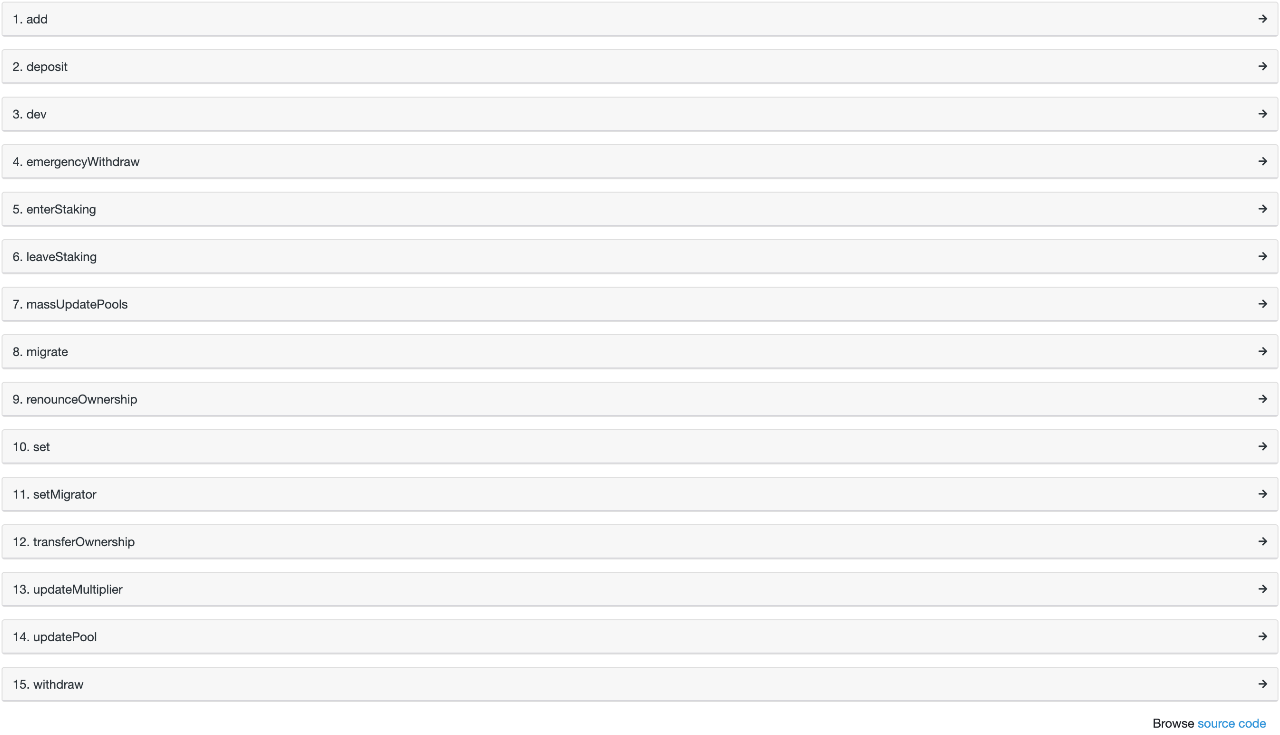
Syrup.sol



1. approve →
2. burn →
3. decreaseAllowance →
4. delegate →
5. delegateBySig →
6. increaseAllowance →
7. mint →
8. mint →
9. renounceOwnership →
10. safeCakeTransfer →
11. transfer →
12. transferFrom →
13. transferOwnership →

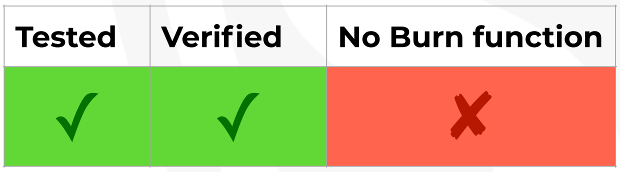
Browse source code

MasterChef.sol



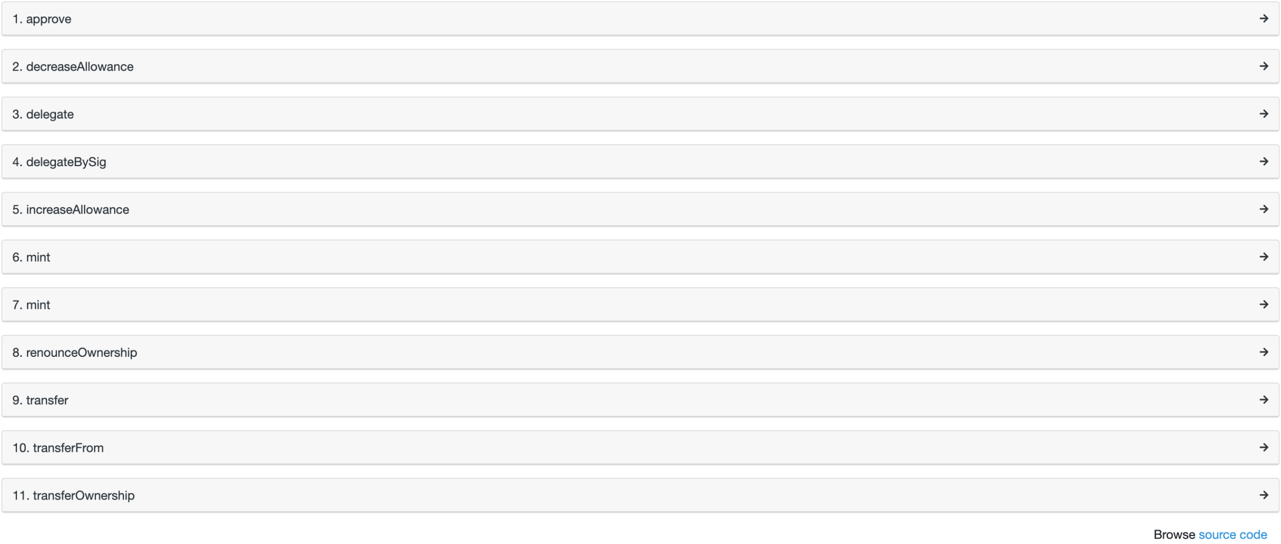
1. add →
2. deposit →
3. dev →
4. emergencyWithdraw →
5. enterStaking →
6. leaveStaking →
7. massUpdatePools →
8. migrate →
9. renounceOwnership →
10. set →
11. setMigrator →
12. transferOwnership →
13. updateMultiplier →
14. updatePool →
15. withdraw →

Browse source code



Tested	Verified	No Burn function
✓	✓	✗

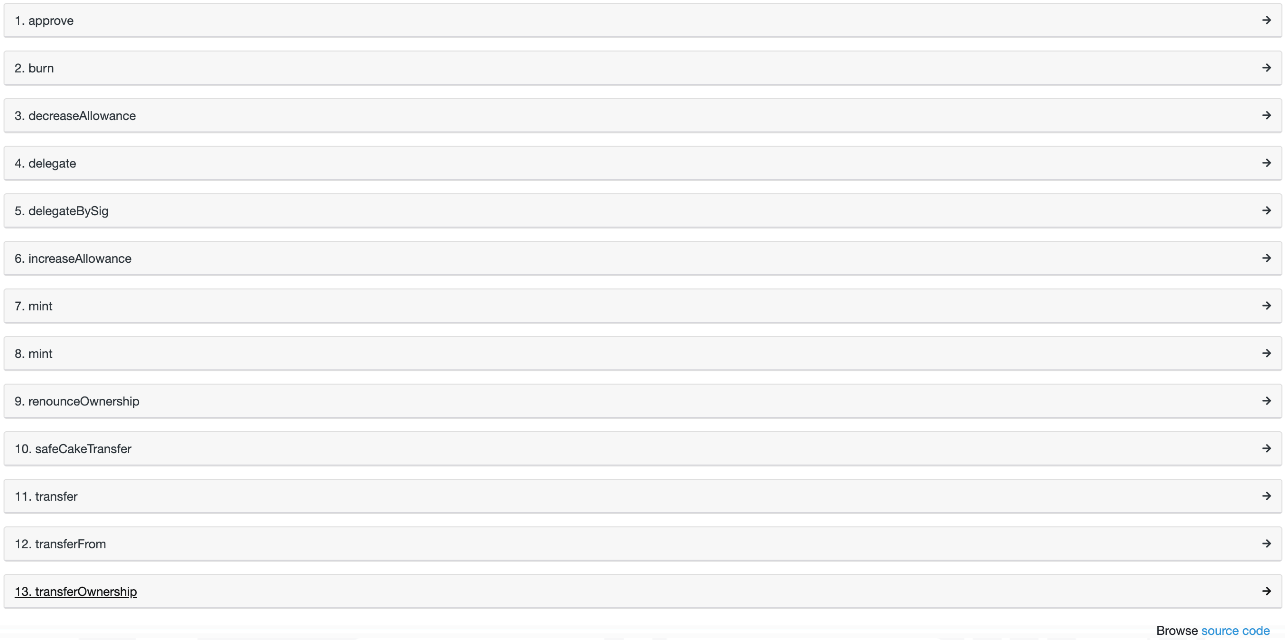
CoffeeMaker.sol



1. approve →
2. decreaseAllowance →
3. delegate →
4. delegateBySig →
5. increaseAllowance →
6. mint →
7. mint →
8. renounceOwnership →
9. transfer →
10. transferFrom →
11. transferOwnership →

Browse source code

Syrup.sol



1. approve →

2. burn →

3. decreaseAllowance →

4. delegate →

5. delegateBySig →

6. increaseAllowance →

7. mint →

8. mint →

9. renounceOwnership →

10. safeCakeTransfer →

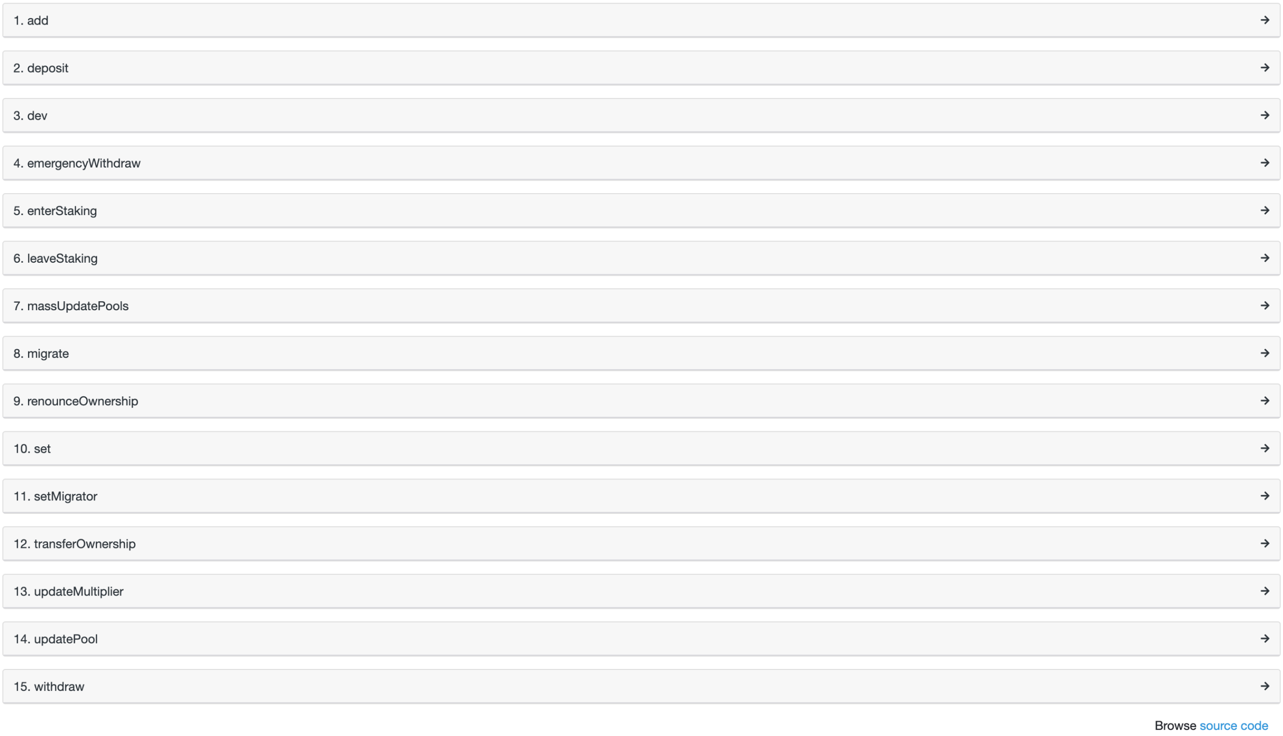
11. transfer →

12. transferFrom →

13. transferOwnership →

Browse source code

MasterChef.sol



1. add →

2. deposit →

3. dev →

4. emergencyWithdraw →

5. enterStaking →

6. leaveStaking →

7. massUpdatePools →

8. migrate →

9. renounceOwnership →

10. set →

11. setMigrator →

12. transferOwnership →

13. updateMultiplier →

14. updatePool →

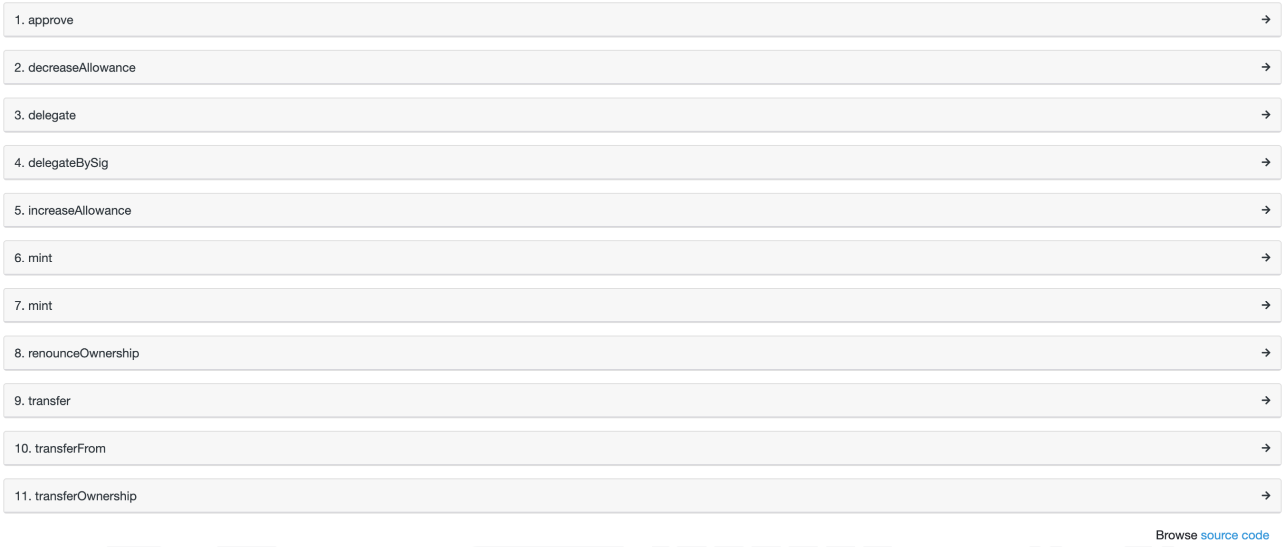
15. withdraw →

Browse source code

Deployer cannot pause the contract

Tested	Verified	No pause function
✓	✓	✓

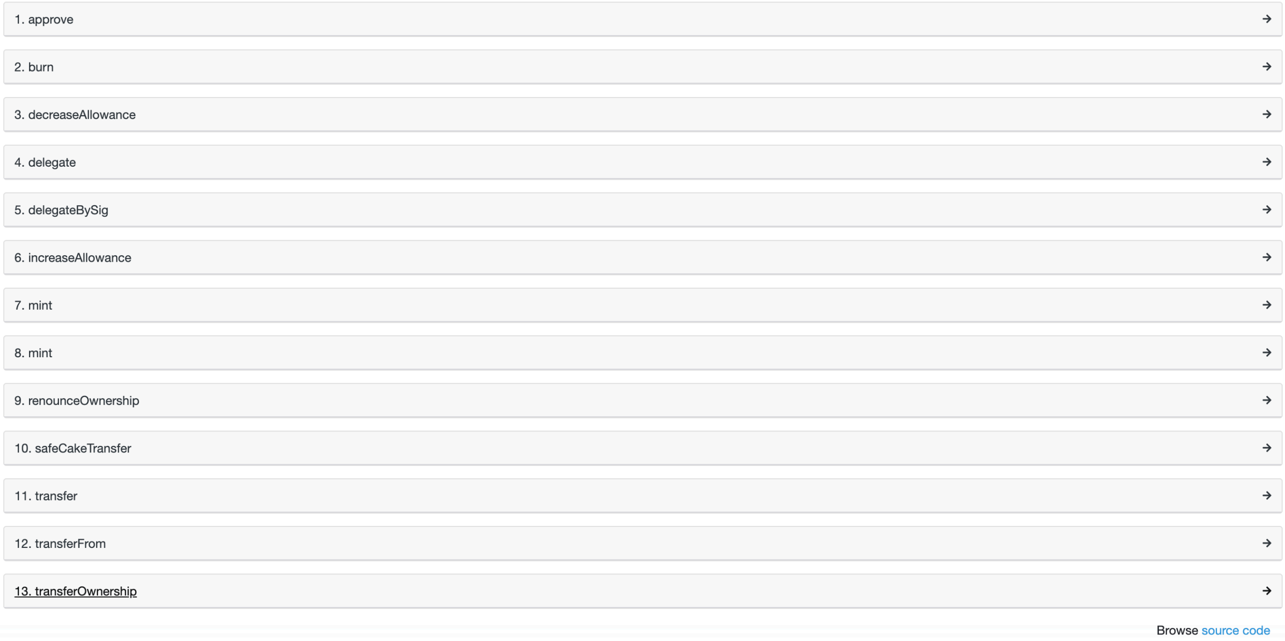
CoffeeMaker.sol



1. approve
2. decreaseAllowance
3. delegate
4. delegateBySig
5. increaseAllowance
6. mint
7. mint
8. renounceOwnership
9. transfer
10. transferFrom
11. transferOwnership

Browse source code

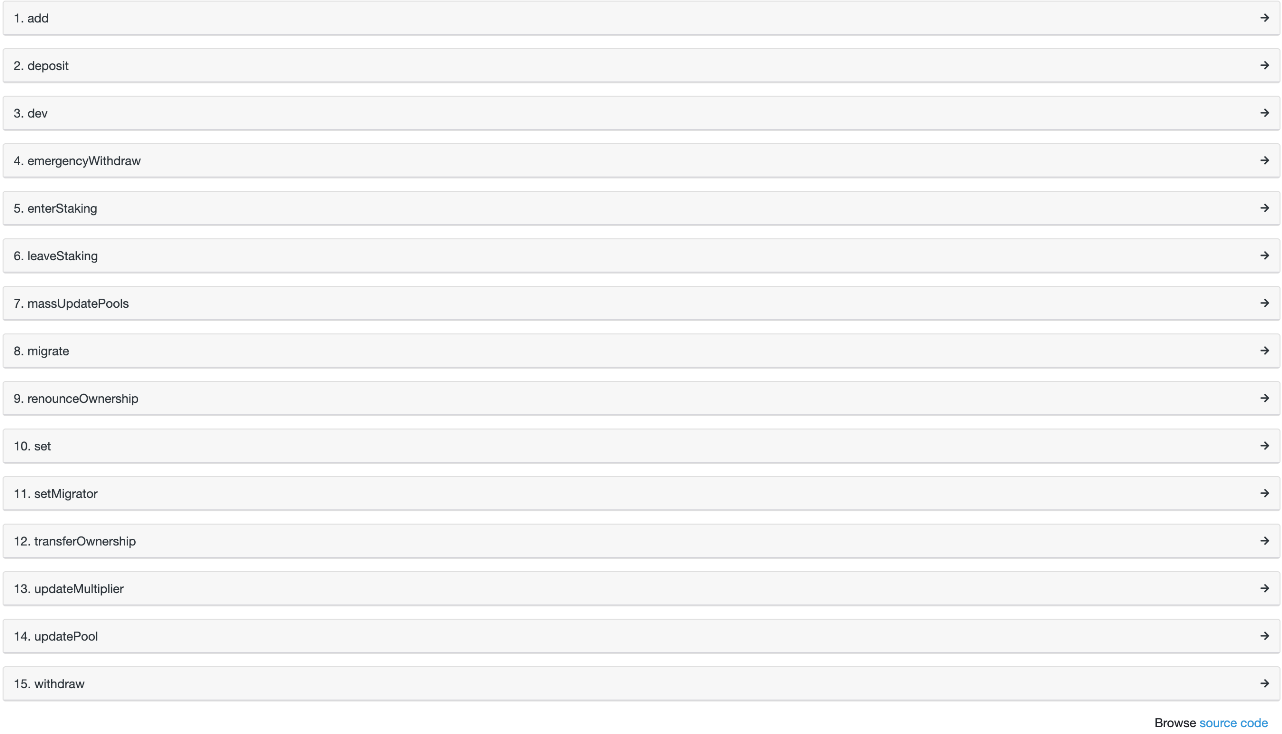
Syrup.sol



1. approve
2. burn
3. decreaseAllowance
4. delegate
5. delegateBySig
6. increaseAllowance
7. mint
8. mint
9. renounceOwnership
10. safeCakeTransfer
11. transfer
12. transferFrom
13. transferOwnership

Browse source code

MasterChef.sol



1. add →
2. deposit →
3. dev →
4. emergencyWithdraw →
5. enterStaking →
6. leaveStaking →
7. massUpdatePools →
8. migrate →
9. renounceOwnership →
10. set →
11. setMigrator →
12. transferOwnership →
13. updateMultiplier →
14. updatePool →
15. withdraw →

Browse source code

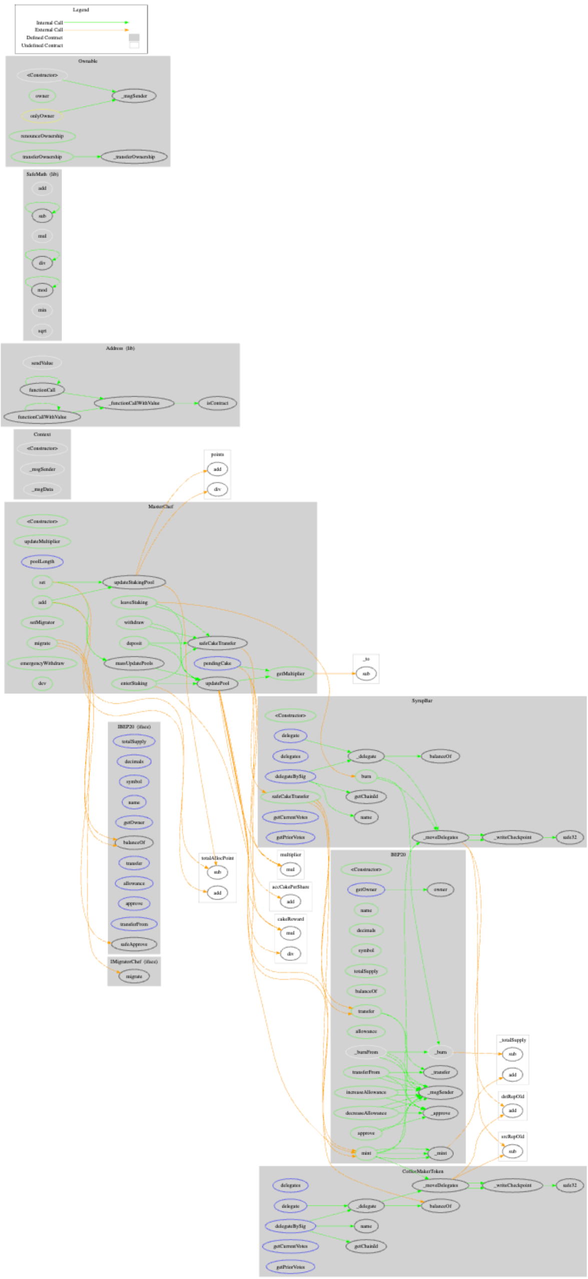
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

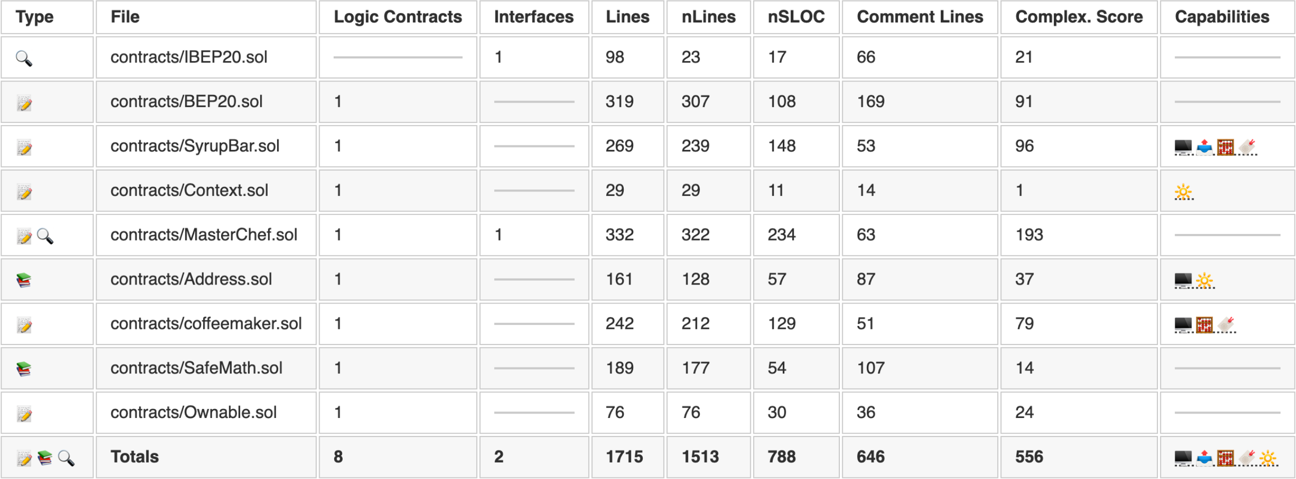
Legend

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

CallGraph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
_msgSender
_transferOwnership
SafeMath (lib)
add
sub
mul
div
mod
min
sqrt
Address (lib)
sendValue
functionCall
functionCallWithValue
_functionCallWithValue
isContract
Context
<Constructor>
_msgSender
_msgData
points
add
div
MasterChef
<Constructor>
updateMultiplier
poolLength
set
add
setMigrator
migrate
emergencyWithdraw
dev
updateStakingPool
leaveStaking
withdraw
deposit
massUpdatePools
enterStaking
safeCakeTransfer
pendingCake
updatePool
getMultiplier
_to
sub
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
safeApprove
IMigratorChef (iface)
migrate
totalAllocPoint
sub
add
SyrupBar
<Constructor>
delegate
delegates
delegateBySig
safeCakeTransfer
getCurrentVotes
getPriorVotes
_delegate
burn
getChainId
name
balanceOf
_moveDelegates
_writeCheckpoint
safe32
multiplier
mul
accCakePerShare
add
cakeReward
mul
div
BEP20
<Constructor>
getOwner
name
decimals
symbol
totalSupply
balanceOf
transfer
allowance
_burnFrom
transferFrom
increaseAllowance
decreaseAllowance
approve
mint
owner
_burn
_transfer
_msgSender
_approve
_mint
_totalSupply
sub
add
dstRepOld
add
srcRepOld
sub
CoffeeMakerToken
delegates
delegate
delegateBySig
getCurrentVotes
getPriorVotes
_delegate
name
getChainId
_moveDelegates
balanceOf
_writeCheckpoint
safe32

Source Units in Scope



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/IBEP20.sol	—	1	98	23	17	66	21	—
📝	contracts/BEP20.sol	1	—	319	307	108	169	91	—
📝	contracts/SyrupBar.sol	1	—	269	239	148	53	96	🖥📤🧮🔖
📝	contracts/Context.sol	1	—	29	29	11	14	1	🔆
📝🔍	contracts/MasterChef.sol	1	1	332	322	234	63	193	—
📚	contracts/Address.sol	1	—	161	128	57	87	37	🖥🔆
📝	contracts/coffeemaker.sol	1	—	242	212	129	51	79	🖥🧮🔖
📚	contracts/SafeMath.sol	1	—	189	177	54	107	14	—
📝	contracts/Ownable.sol	1	—	76	76	30	36	24	—
📝📚🔍	**Totals**	**8**	**2**	**1715**	**1513**	**788**	**646**	**556**	🖥📤🧮🔖🔆

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

Issue	File	Type	Line	Description
#1	MasterChef	Reentrancy vulnerabilities (no theft of ethers) (reentrancy-no-eth)	313-320	Reentrancy in MasterChef.emergencyWithdraw (uint256): •pool.lpToken.safeTransfer(address(msg.sender),user.amount) • user.amount = 0 • user.rewardDebt = 0

#2	MasterChef	Reentrancy vulnerabilities (no theft of ethers) (reentrancy-no-eth)	229-248	Reentrancy in MasterChef.deposit(uint256,uint256): · updatePool(_pid) · cake.mint(devaddr,cakeReward.div(10)) · cake.mint(address(syrup),cakeReward) · safeCakeTransfer(msg.sender,pending) · syrup.safeCakeTransfer(_to,_amount) · pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) · user.amount = user.amount.add(_amount) · user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12)

Low issues

Issue	File	Type	Line	Description
#1	Main	Potential use of "block.number" as source of randonmness	147-149, 220-222	The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

#2	Main	Loop over unbounded data structure.	167	Gas consumption in function "getPriorVotes" in contract "CoffeeMakerToken" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.
#3	Main	A control flow decision is made based on The block.timestamp environment variable.	117	The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Informational issues

- no informational issues found -

Commented Code exist

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

Line	File	Comment
11	MasterChef.sol	// import "@nomiclabs/buidler/console.sol";

Recommendation

Remove the commented code, or address them properly.

Audit Comments

24. July 2021, MasterChef.sol: There is still an owner (Owner still has not renounced ownership, address: 0x7dadc121c38cf76fd70e8babd846435125e935cc)

24. July 2021, SyrupBar.sol: There is still an owner (Owner still has not renounced ownership, address: 0x7dadc121c38cf76fd70e8babd846435125e935cc)

24. July 2021, CoffeeMaker.sol: There is still an owner (Owner still has not renounced ownership, address: 0x7dadc121c38cf76fd70e8babd846435125e935cc)

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	NOT 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	NOT 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	NOT 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	NOT 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**

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY