SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

29. January, 2022

For

Beli Finance

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

Network
Binance Smart Chain (BEP20)

Website
https://beli.finance/

Description

Beli Finance is a decentralized reserve policy-controlled currency protocol, Multi-Chain Yield Optimizer Aggregator platform that convert yield farm into $BELI token and still earn small compound interest on their crypto holdings automatically. Each $BELI token is backed by a basket of assets (e.g., USDT, BNB, BUSD Tokens etc etc) in the treasury, giving it an intrinsic value that it cannot fall below. Beli Finance also introduces economic and game-theoretic dynamics into the market through staking and compounding.

Project Engagement

During the 27th of January 2022, **Beli Finance 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

- Github
  - https://github.com/belifinance/beli-finance
  - Commit: fe03b705871f742e0eabe3cad79ba9e75c781ff6

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:

```
./helpers/ERC20.sol
./libraries/Address.sol
./libraries/SafeERC20.sol
./helpers/Ownable.sol
```

Tested Contract Files

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

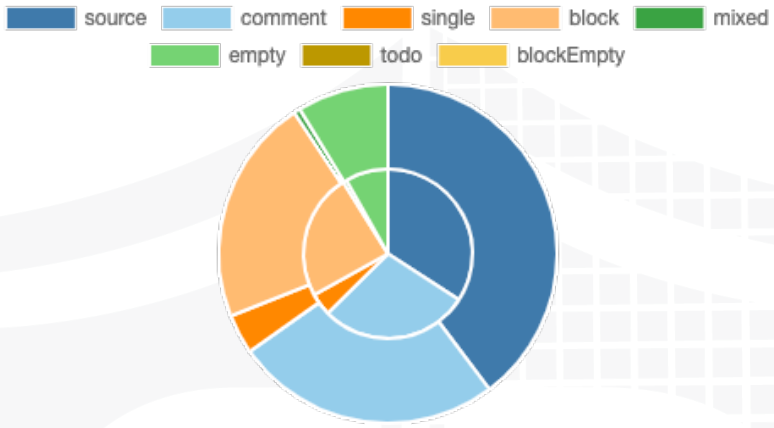
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

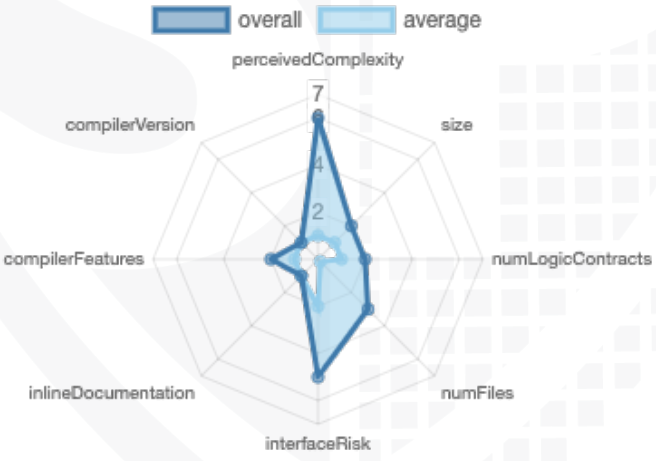
File Name	SHA-1 Hash
contracts/LiquidityLocker.sol	6c288a231f056ddbf9561de798f8ef31d9c67823
contracts/interfaces/IFarm.sol	1f113afd8084573e154fd87243932a453534d9cc
contracts/interfaces/IFeeReceiver.sol	59494838b301b4fe7a245c7daa2060a4a287bc83
contracts/interfaces/ILocker.sol	164a536f14b263d8f219391c884890d9655d01e2
contracts/interfaces/IPancakeswapFarm.sol	70329126e7295a7e648c0a40755f8c48a27d7517
contracts/interfaces/IReferral.sol	ceff967b0354f9259cd458a66b0bfed5b49daf15
contracts/interfaces/IPancakeRouter02.sol	464cfb67e696411142cf03786eea8d5bb60cfbfe
contracts/interfaces/IPancakeRouter01.sol	d36e679a84938d5bd06c2548ba13d47f7e22f81d
contracts/interfaces/IVault.sol	e90638e1d89bcf47aa306faef410e2110811f1f5
contracts/interfaces/IWBNB.sol	adf33802d70f1b5660b1c4523bac19bbeaca6d8a
contracts/interfaces/IERC20.sol	347c58c28cbce34e2d6376f870c7ac45d8d82400
contracts/helpers/AccessControl.sol	5eb32cb05cdab14507bce2ca40ef20d42830cad5
contracts/helpers/Context.sol	b9599e1bf4c3eff19e61490e31db3a18ccb72dd3
contracts/helpers/Ownable.sol	72716c08c4af60b8e10550b94dfb348bd377e02e
contracts/helpers/Pausable.sol	8bcd3e3e43173dd4b7576eb131cfe81990ad2aa0
contracts/helpers/ERC20.sol	8d193f5737a404caa24c1f3817231c74e4078249
contracts/helpers/ReentrancyGuard.sol	2fa15db6f6bc0f1822b3658a60fa471e3ae3c624
contracts/libraries/EnumerableSet.sol	c206160e3aa76ce5e6d5394f2a3367491aa61dd8
contracts/libraries/Address.sol	32626bab7e8068c3c5f578532d3222771d17647d
contracts/libraries/SafeMath.sol	c1193bc1ea44695594726881e36783e641eb5214
contracts/libraries/SafeERC20.sol	24a84d83d8bd2015df22c14f512edf0b5e01bcda
contracts/BeliToken.sol	2929f4a111a10afcf71bc6642880a1ce65836a95

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	4	4	10	4

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	123	5

Version	External	Internal	Private	Pure	View
1.0	90	166	10	14	56

State Variables

Version	Total	Public
1.0	35	22

Capabilities

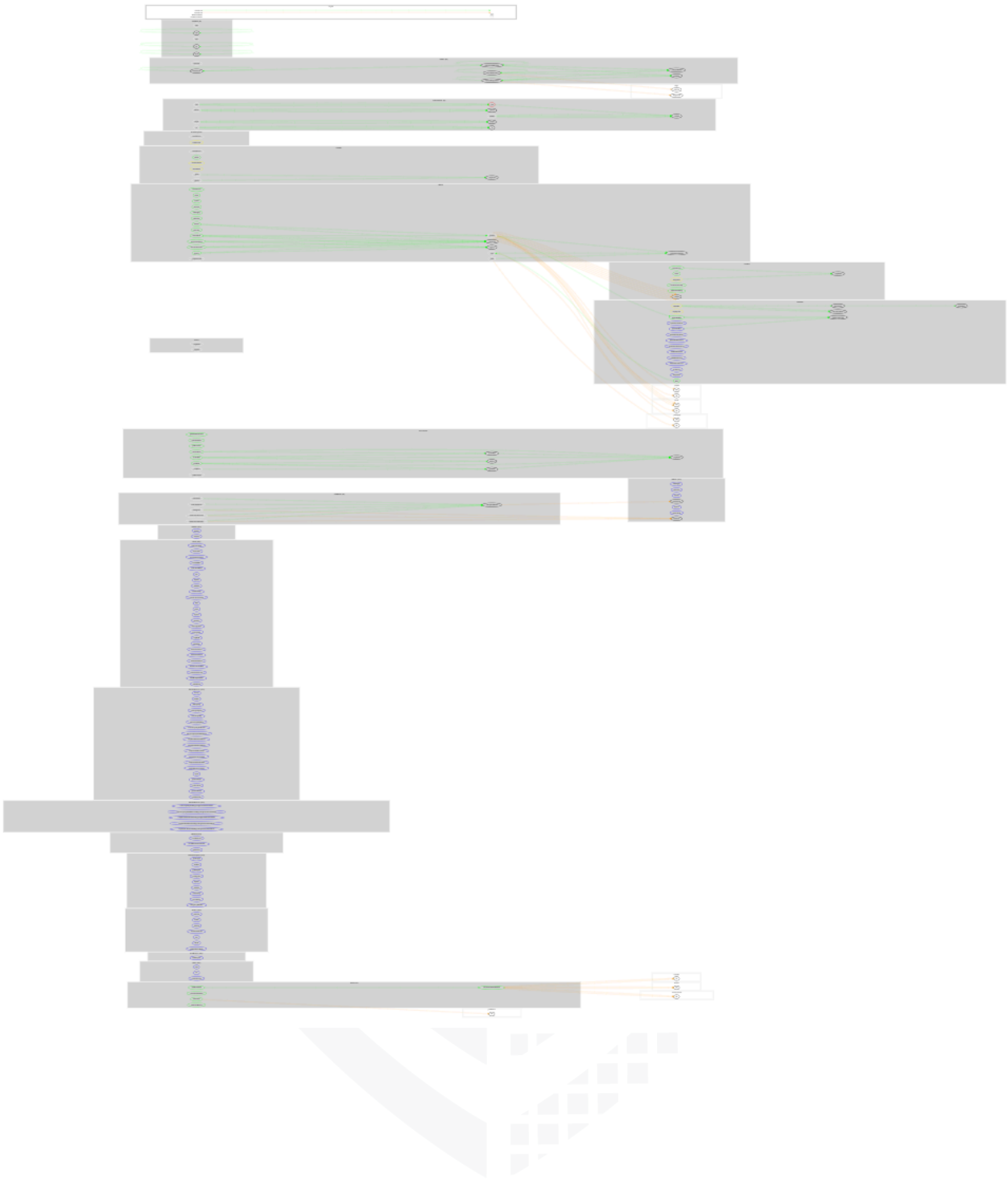
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`0.6.12` `^0.6.12`		`yes`	`yes` (2 asm blocks)	

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

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. 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)

Correct implementation of Token standard

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

updateTransferLimit
updateFeeRate
updateTreasuryFactor
updateBeliStakeFactor
updateBeliLPStakeFactor
setWhaleExclusion
setBeliFeeReceiver
setBeliLPFeeReceiver
setTreasury
setOperator
mint

transfer
approve
transferFrom
increaseAllowance
decreaseAllowance

renounceOwnership
transferOwnership

Deployer cannot mint any new tokens

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

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Tokens will burn while transfer

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

Overall checkup (Smart Contract Security)

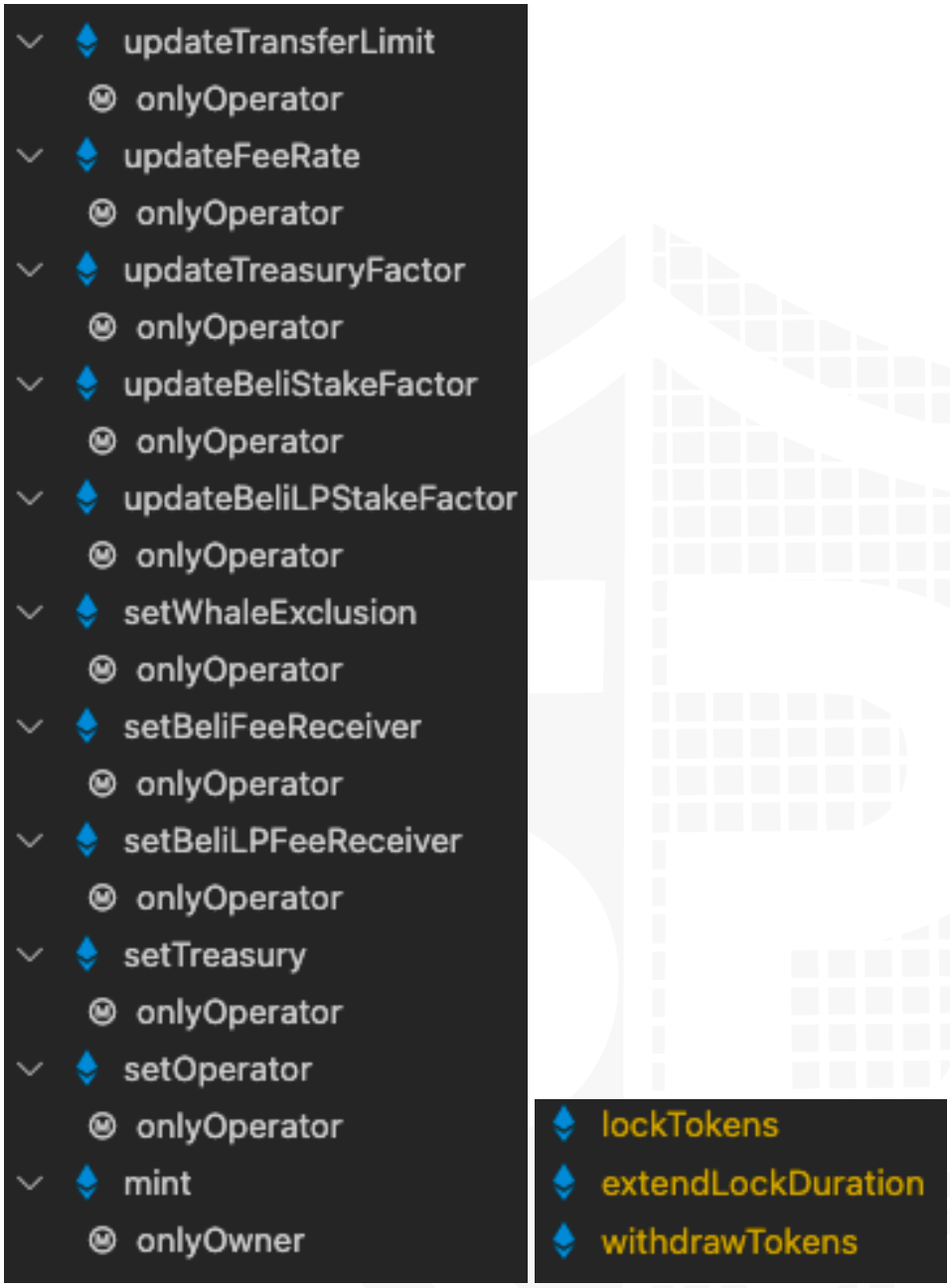
Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verfified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	-

Modifiers and public functions

v1.0



Comments

- Deployer can mint new tokens without limitations

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/LiquidityLocker.sol	1	———	155	150	105	27	78	🔹
🔍	contracts/interfaces/IFarm.sol	———	1	40	12	6	4	7	———
🔍	contracts/interfaces/IFeeReceiver.sol	———	1	6	5	3	1	3	———
🔍	contracts/interfaces/ILocker.sol	———	1	18	5	3	1	15	———
🔍	contracts/interfaces/IPancakeswapFarm.sol	———	1	34	4	3	7	19	———
🔍	contracts/interfaces/IReferral.sol	———	1	20	9	3	10	7	———
🔍	contracts/interfaces/IPancakeRouter02.sol	———	1	50	6	4	———	16	💰
🔍	contracts/interfaces/IPancakeRouter01.sol	———	1	160	4	3	———	48	💰
🔍	contracts/interfaces/IVault.sol	———	1	75	6	3	9	51	———
🔍	contracts/interfaces/IWBNB.sol	———	1	10	7	4	1	10	💰
🔍	contracts/interfaces/IERC20.sol	———	1	86	26	21	54	13	🔆
🎨	contracts/helpers/AccessControl.sol	1	———	206	202	79	101	43	🔆
🎨	contracts/helpers/Context.sol	1	———	13	13	10	2	1	🔆
🎨	contracts/helpers/Ownable.sol	1	———	62	62	33	21	23	———
📝	contracts/helpers/Pausable.sol	1	———	80	80	29	41	14	🔆
📝	contracts/helpers/ERC20.sol	1	———	345	305	113	161	81	———
🎨	contracts/helpers/ReentrancyGuard.sol	1	———	45	45	15	22	5	———
📚	contracts/libraries/EnumerableSet.sol	1	———	321	274	98	141	34	🔆
📚	contracts/libraries/Address.sol	1	———	248	180	93	110	47	💻👥
📚	contracts/libraries/SafeMath.sol	1	———	157	145	39	93	10	🔆
📚	contracts/libraries/SafeERC20.sol	1	———	123	98	66	23	25	🔆
📝	contracts/BeliToken.sol	1	———	199	199	161	22	130	———
📝📚🔍🎨	**Totals**	**12**	**10**	**2453**	**1837**	**894**	**851**	**680**	💻💰🔹👥🔆

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

Issue	File	Type	Line	Description
#1	BeliToken	The arithmetic operator can overflow.	68, 71, 74	It is possible to cause an integer overflow or underflow in the arithmetic operation. We recommend you to use SafeMath Library operations

Medium issues

Issue	File	Type	Line	Description
#1	LiquidityLocker	Unchecked tokens transfer	148	Use `SafeERC20`, or ensure that the transfer/ transferFrom return value is checked

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	BeliToken	Missing Zero Address Validation (missing-zero-check)	127, 132, 142, 137	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	All Helper files	Missing SPDX License	-	Add SPDX License to source file
#2	IERC20	Missing SPDX License	-	Add SPDX License to source file
#3	All Pancake interfaces	Missing SPDX License	-	Add SPDX License to source file

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/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.

29. January 2022:

- 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	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

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	NOT PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY