SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

17. August, 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	10. August 2021	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	17. August 2021	• Recheck changes

Network
Binance Smart Chain (BEP20)

Website
https://bb.sale/

Telegram
https://t.me/bbsaletoken

Twitter
https://twitter.com/cryptobbsale

Description

Project Engagement

During the 09 of August 2021, **BBSale 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. **BBSale Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

TBA

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
    - SafeMath
- Uniswap
    - UniswapV2Factory
    - UniswapV2Pair
    - UniswapV2Router01
    - UniswapV2Router02

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/PerSale/import/PreSaleInfo.sol	7b77b48025f7c8908488c64d652ccaefb89501ef
contracts/PerSale/CreatePreSale.sol	00f781489f83997cefd1711323403d924d5dadb1
contracts/Lock/LockToken.sol	954fc491056f294395cc2e7024894fca6d37865f
contracts/Lock/LockLPToken.sol	e414401494037b51099a563163a2fa2579d8d988
contracts/MintToken/import/liquidityToken.sol	a361c57d9ea5f1910c51f6e091393e8ac819aea6
contracts/MintToken/import/standardToken.sol	e59fe0f4e9f071ef023a080f3a622af3275eccaa
contracts/MintToken/MintLiquidityToken.sol	5c0bd2c04c901ad6eb4a438ae4501fa97623970d
contracts/MintToken/MintStandardToken.sol	aeb1caf70cacad67d0facf23f1625a82cb4812ef

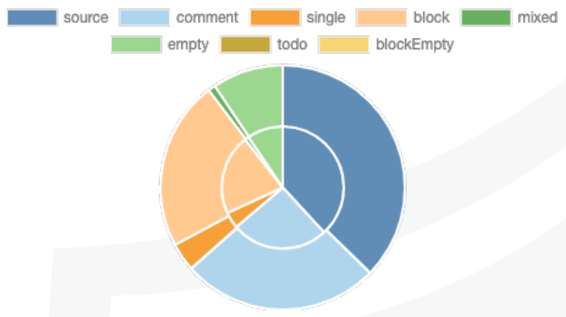
v1.1

File Name	SHA-1 Hash
contracts/PerSale/import/PreSaleInfo.sol	5fcdd8661bc2b6d11b2b6da446d979a1a0c31b18
contracts/PerSale/CreatePreSale.sol	ab5bd7ba94d3a414015dbb4508cf0c2d7fec4d73
contracts/Lock/LockToken.sol	a6b79532a21677a5bf7b3c5fad692397cc256421
contracts/Lock/LockLPToken.sol	b04e4c8be2edc8b90ee23f2dde7f6b61d9214d75
contracts/MintToken/import/liquidityToken.sol	8885011f9f1fa82c9ab39af6e911735e02b7bf70
contracts/MintToken/import/standardToken.sol	a5ce2f587a7af3258ba86eedd7f94fa174c0fa5c
contracts/MintToken/MintLiquidityToken.sol	437f772ab3672e5b172fab5cdcf28b92522fef37
contracts/MintToken/MintStandardToken.sol	c88049b4339eacafe7385efa8fceaa4d18be6e75

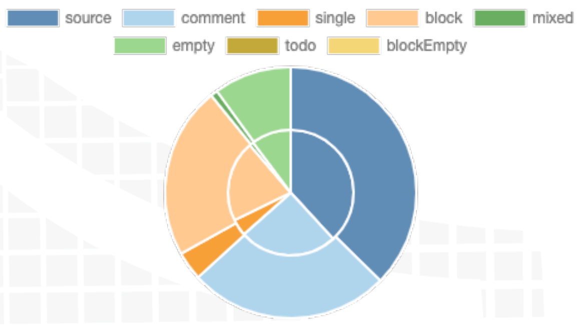
Metrics

Source Lines

v1.0

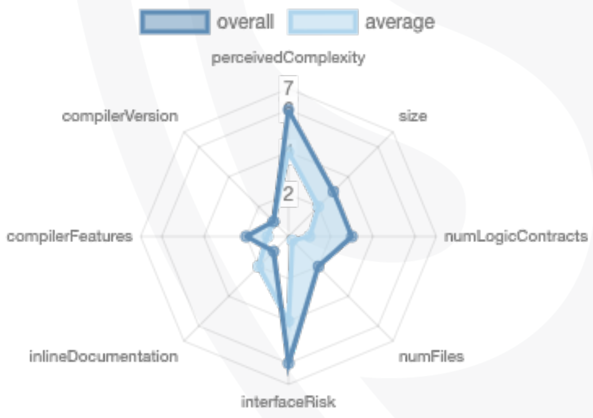


v1.1

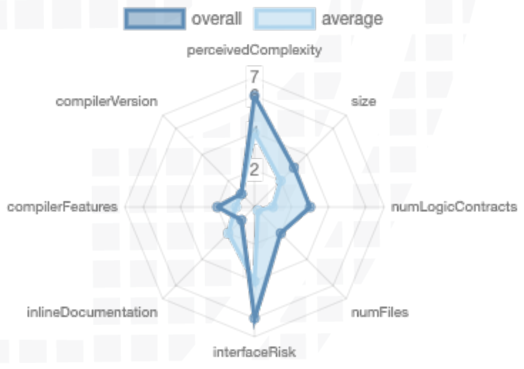


Risk Level

v1.0



v1.1



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	13	9	17	5
1.1	13	9	17	5

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	255	17
1.1	260	17

Version	External	Internal	Private	Pure	View
1.0	174	263	27	63	113
1.1	174	267	28	64	113

State Variables

Version	Total	Public
1.0	125	87
1.1		

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`^0.8.4`	`ABIEncoderV2`	`yes`	`yes` (8 asm blocks)	

Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/ Create/ Create2
`yes`					`yes` → `NewContract:PreSaleInfo` → `NewContract:LiquidityGeneratorToken` → `NewContract:StandardToken`

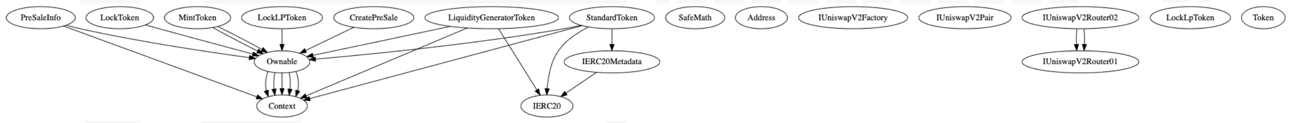
Scope of Work

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

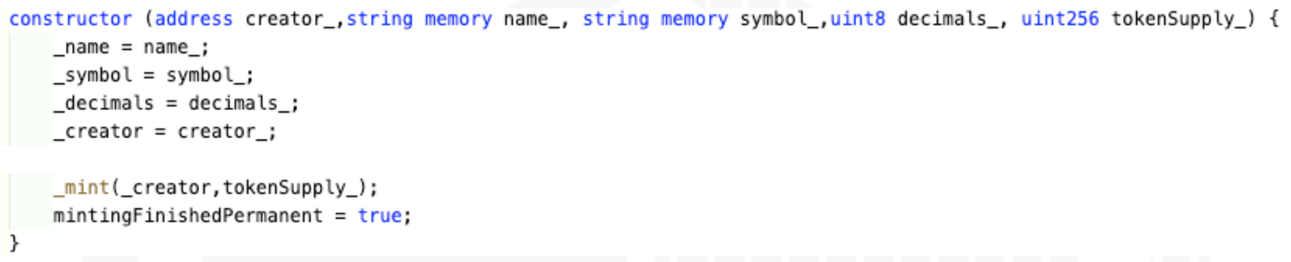
Function	Description	Exist	Tested	Verified
renounceOwnership	Owner renounce ownership for more trust	✓	✓	✓

Deployer cannot mint any new tokens

Tested	Deployer cannot mint	File	Comment
✓	✓	Main	Line: -

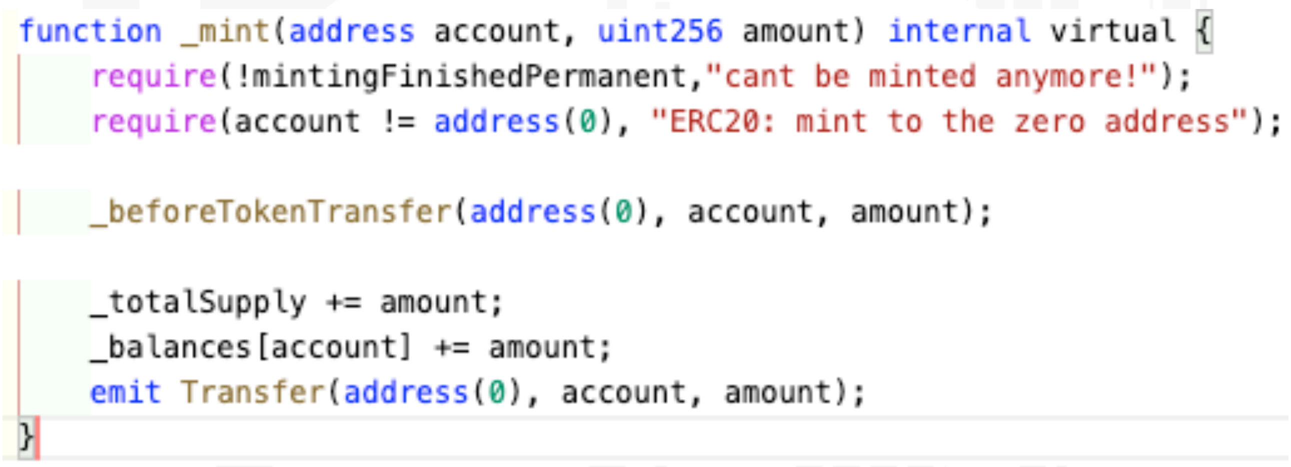
Max / Total Supply: Total supply can be set by owner while deploying

StandardToken



```
constructor (address creator_,string memory name_, string memory symbol_,uint8 decimals_, uint256 tokenSupply_) {
    _name = name_;
    _symbol = symbol_;
    _decimals = decimals_;
    _creator = creator_;

    _mint(_creator,tokenSupply_);
    mintingFinishedPermanent = true;
}
```

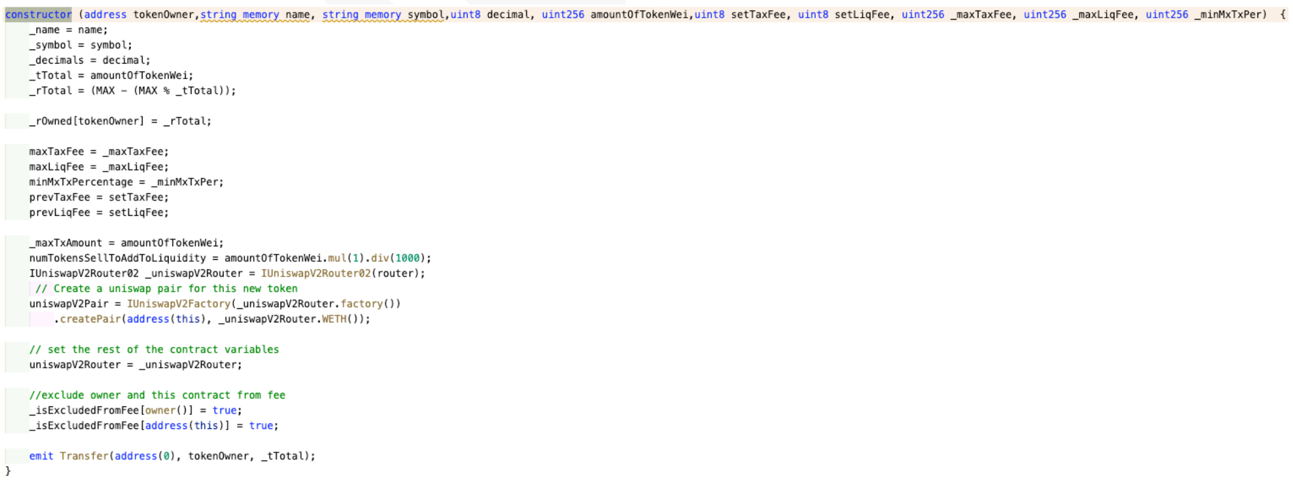


```
function _mint(address account, uint256 amount) internal virtual {
    require(!mintingFinishedPermanent,"cant be minted anymore!");
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

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

LiquidityToken



```
constructor (address tokenOwner,string memory name, string memory symbol,uint8 decimal, uint256 amountOfTokenWei,uint8 setTaxFee, uint8 setLiqFee, uint256 _maxTaxFee, uint256 _maxLiqFee, uint256 _minMxTxPer)  {
    _name = name;
    _symbol = symbol;
    _decimals = decimal;
    _tTotal = amountOfTokenWei;
    _rTotal = (MAX - (MAX % _tTotal));

    _rOwned[tokenOwner] = _rTotal;

    maxTaxFee = _maxTaxFee;
    maxLiqFee = _maxLiqFee;
    minMxTxPercentage = _minMxTxPer;
    prevTaxFee = setTaxFee;
    prevLiqFee = setLiqFee;

    _maxTxAmount = amountOfTokenWei;
    numTokensSellToAddToLiquidity = amountOfTokenWei.mul(1).div(1000);
    IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(router);
     // Create a uniswap pair for this new token
    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

    // set the rest of the contract variables
    uniswapV2Router = _uniswapV2Router;

    //exclude owner and this contract from fee
    _isExcludedFromFee[owner()] = true;
    _isExcludedFromFee[address(this)] = true;

    emit Transfer(address(0), tokenOwner, _tTotal);
}
```

Deployer cannot burn or lock user funds

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

Deployer cannot pause the contract

Tested	Verified	Deployer cannot pause
✓	✓	✓

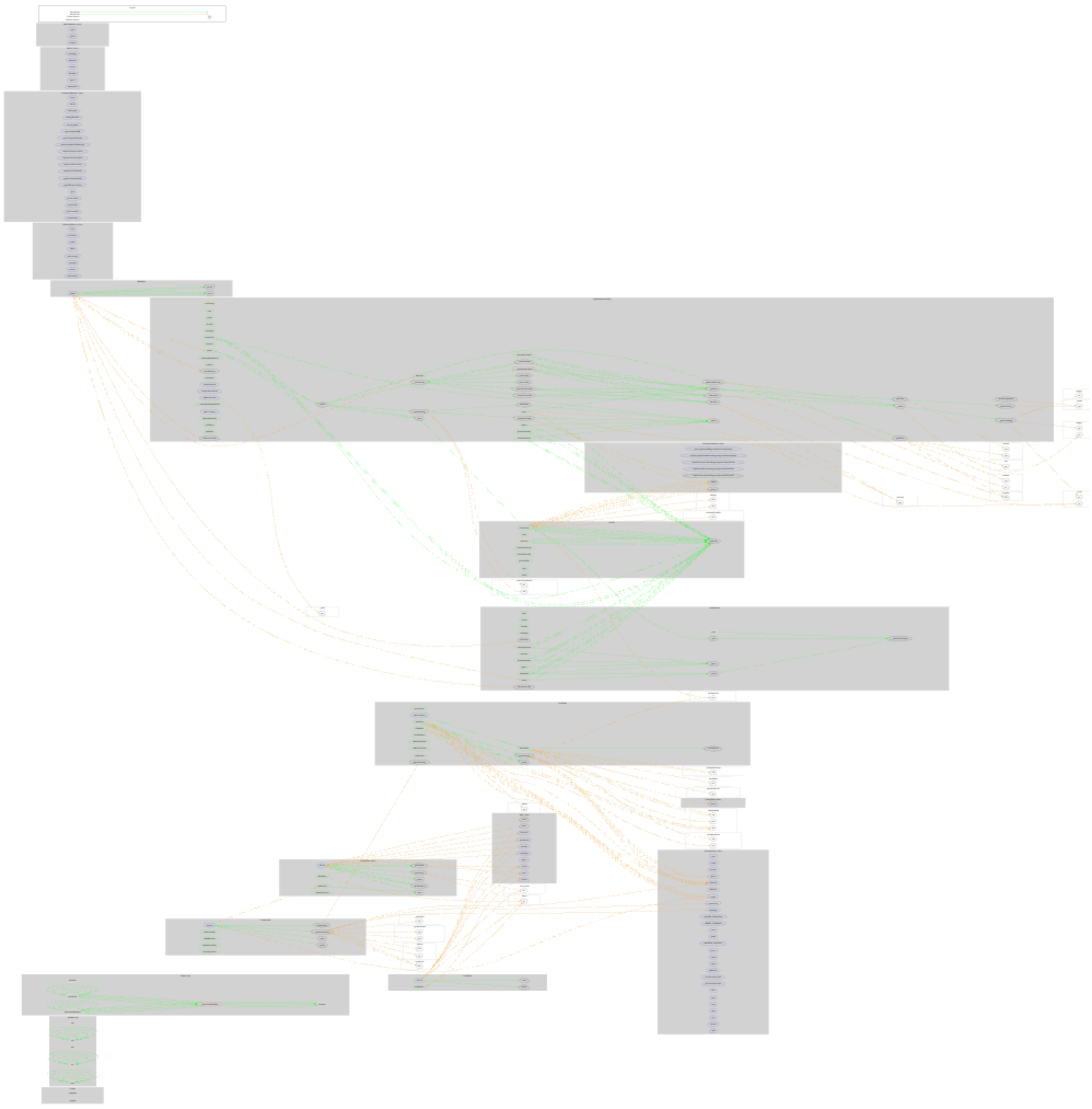
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

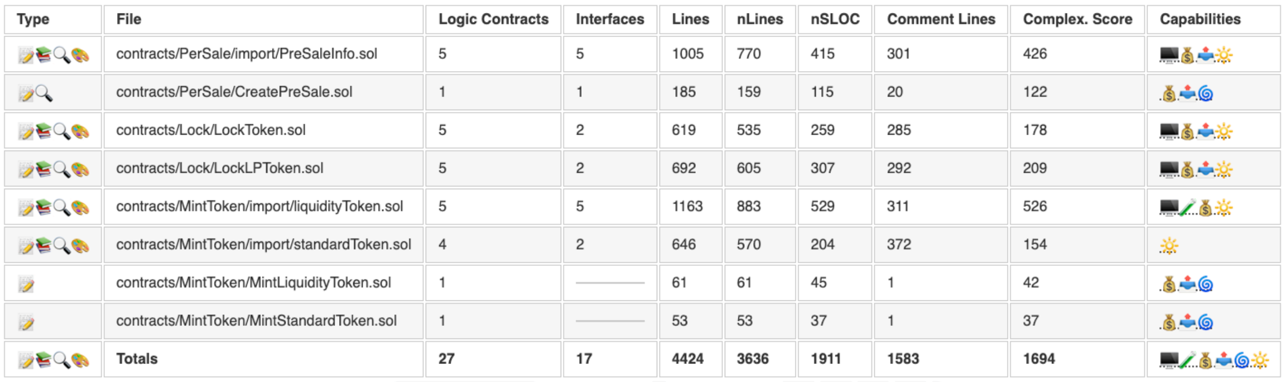
Legend

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

CallGraph



Source Units in Scope



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	contracts/PerSale/import/PreSaleInfo.sol	5	5	1005	770	415	301	426	💻💰📤🔆
📝🔍	contracts/PerSale/CreatePreSale.sol	1	1	185	159	115	20	122	💰📤🌀
📝📚🔍🎨	contracts/Lock/LockToken.sol	5	2	619	535	259	285	178	💻💰📤🔆
📝📚🔍🎨	contracts/Lock/LockLPToken.sol	5	2	692	605	307	292	209	💻💰📤🔆
📝📚🔍🎨	contracts/MintToken/import/liquidityToken.sol	5	5	1163	883	529	311	526	💻🧪💰🔆
📝📚🔍🎨	contracts/MintToken/import/standardToken.sol	4	2	646	570	204	372	154	🔆
📝	contracts/MintToken/MintLiquidityToken.sol	1	----	61	61	45	1	42	💰📤🌀
📝	contracts/MintToken/MintStandardToken.sol	1	----	53	53	37	1	37	💰📤🌀
📝📚🔍🎨	**Totals**	**27**	**17**	**4424**	**3636**	**1911**	**1583**	**1694**	💻🧪💰📤🌀🔆

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	PreSaleInfo	Reentrancy vulnerabilities (theft of ethers) (reentrancy-eth)	808-831, 834-882	Move state variables before executing the transfer itself to avoid reentrance into the function again.

Low issues

Issue	File	Type	Line	Description
#1	LiquidityToken	State variable visibility is not set	686, 719	It is best practice to set the visibility of state variables explicitly.
#2	MintLiquidityToken	State variable visibility is not set	22	It is best practice to set the visibility of state variables explicitly

Informational issues

- no informational issues found -

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	NOT 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