SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Souls of Meta

Audit

Security Assessment
16. April, 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	15. April 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Binance Smart Chain (BEP20)
Polygon Matic

Website

https://soulsofmeta.io/

Telegram

https://t.me/SoulsOfMetaOfficial

Twitter

https://twitter.com/SoulsOfMeta

Facebook

https://www.facebook.com/SoulsOfMeta

Medium

https://soulsofmeta.medium.com/

Youtube

https://www.youtube.com/channel/UCLVNKgHfKRt6DpagPgJSzwA?sub_confirmation=1

Description

SOULS OF MΞTΛ IS A CROSS-GAME MULTI-CHAIN FUN-2-EARN 3RD-PERSON ACTION RPG FANTASY NFT GAMING METAVERSE OF BLADES AND SORCERY
where you can own, play, and monetize NFT assets through GameFi and SocialFi, and travel through community-created realms, fight monsters, collaborate with other players (PvE & PvP), solve quests and beyond, to have fun playing and earn at the same time!

Project Engagement

During the 13th of April 2022, **Soul of Meta 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/SOULS-OF-META/Smart-Contracts
  - Commit: 127004510c8603b82dfab3743a53a90116f244f0

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.

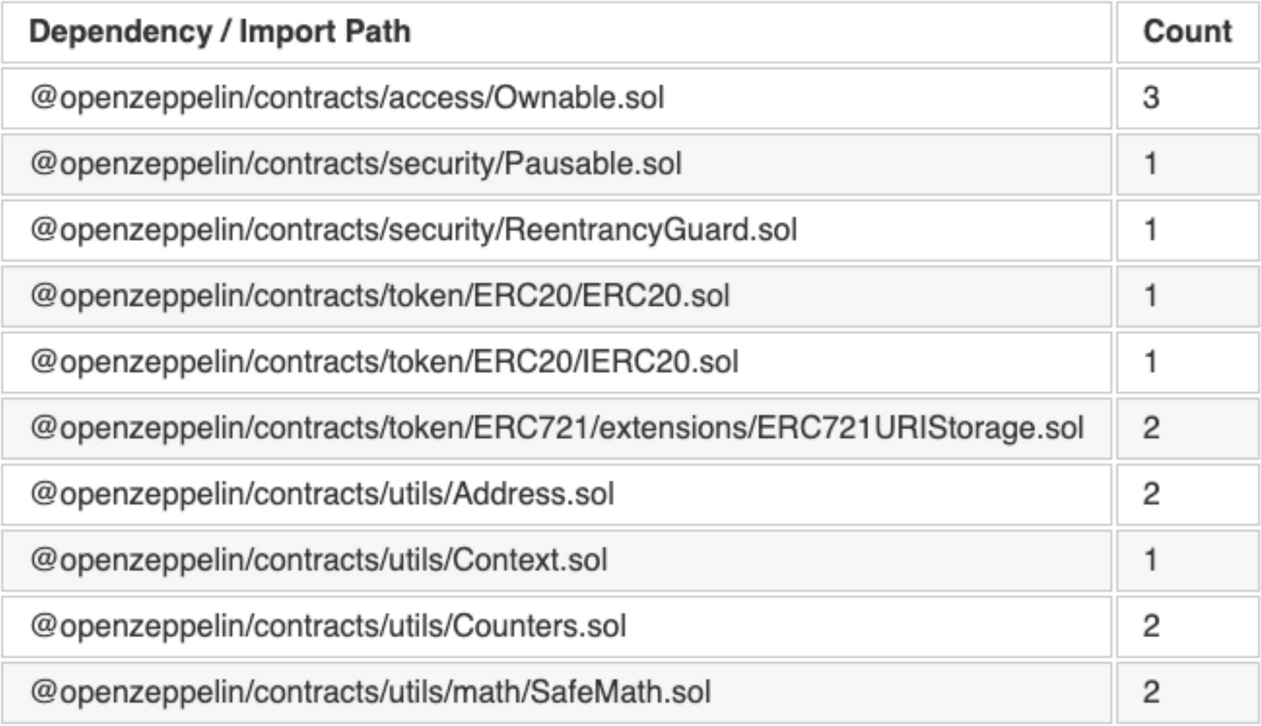
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program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



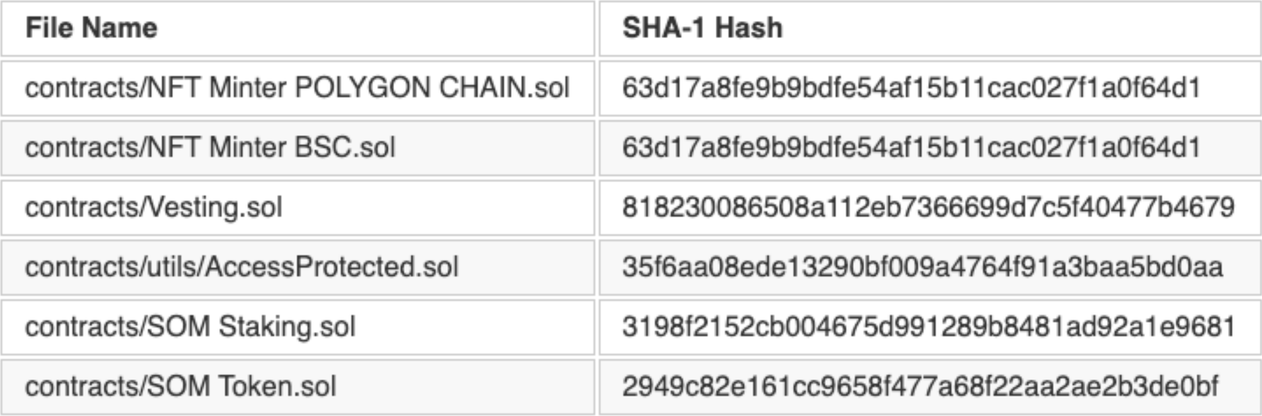
Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	3
@openzeppelin/contracts/security/Pausable.sol	1
@openzeppelin/contracts/security/ReentrancyGuard.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol	2
@openzeppelin/contracts/utils/Address.sol	2
@openzeppelin/contracts/utils/Context.sol	1
@openzeppelin/contracts/utils/Counters.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

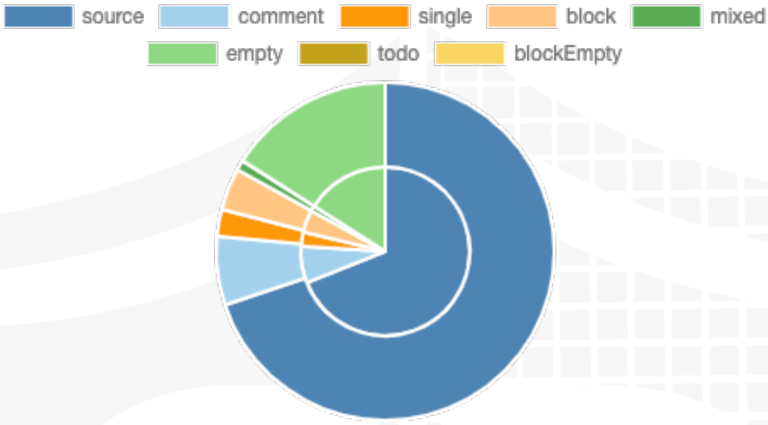


File Name	SHA-1 Hash
contracts/NFT Minter POLYGON CHAIN.sol	63d17a8fe9b9bdfe54af15b11cac027f1a0f64d1
contracts/NFT Minter BSC.sol	63d17a8fe9b9bdfe54af15b11cac027f1a0f64d1
contracts/Vesting.sol	818230086508a112eb7366699d7c5f40477b4679
contracts/utils/AccessProtected.sol	35f6aa08ede13290bf009a4764f91a3baa5bd0aa
contracts/SOM Staking.sol	3198f2152cb004675d991289b8481ad92a1e9681
contracts/SOM Token.sol	2949c82e161cc9658f477a68f22aa2ae2b3de0bf

Metrics

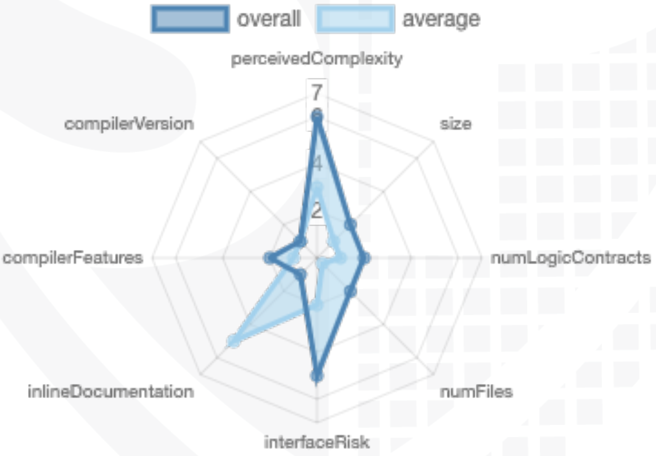
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	5	0	3	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	75	4

Version	External	Internal	Private	Pure	View
1.0	36	70	18	6	27

State Variables

Version	Total	Public
1.0	51	31

Capabilities

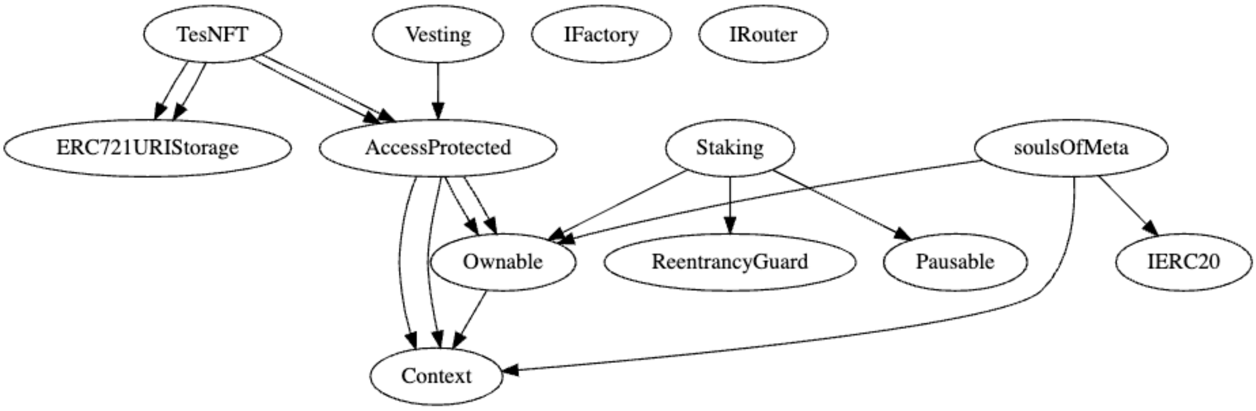
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts

1.0	`^0.8.4`
	`0.8.4`
	`^0.8.7`
			`yes`		

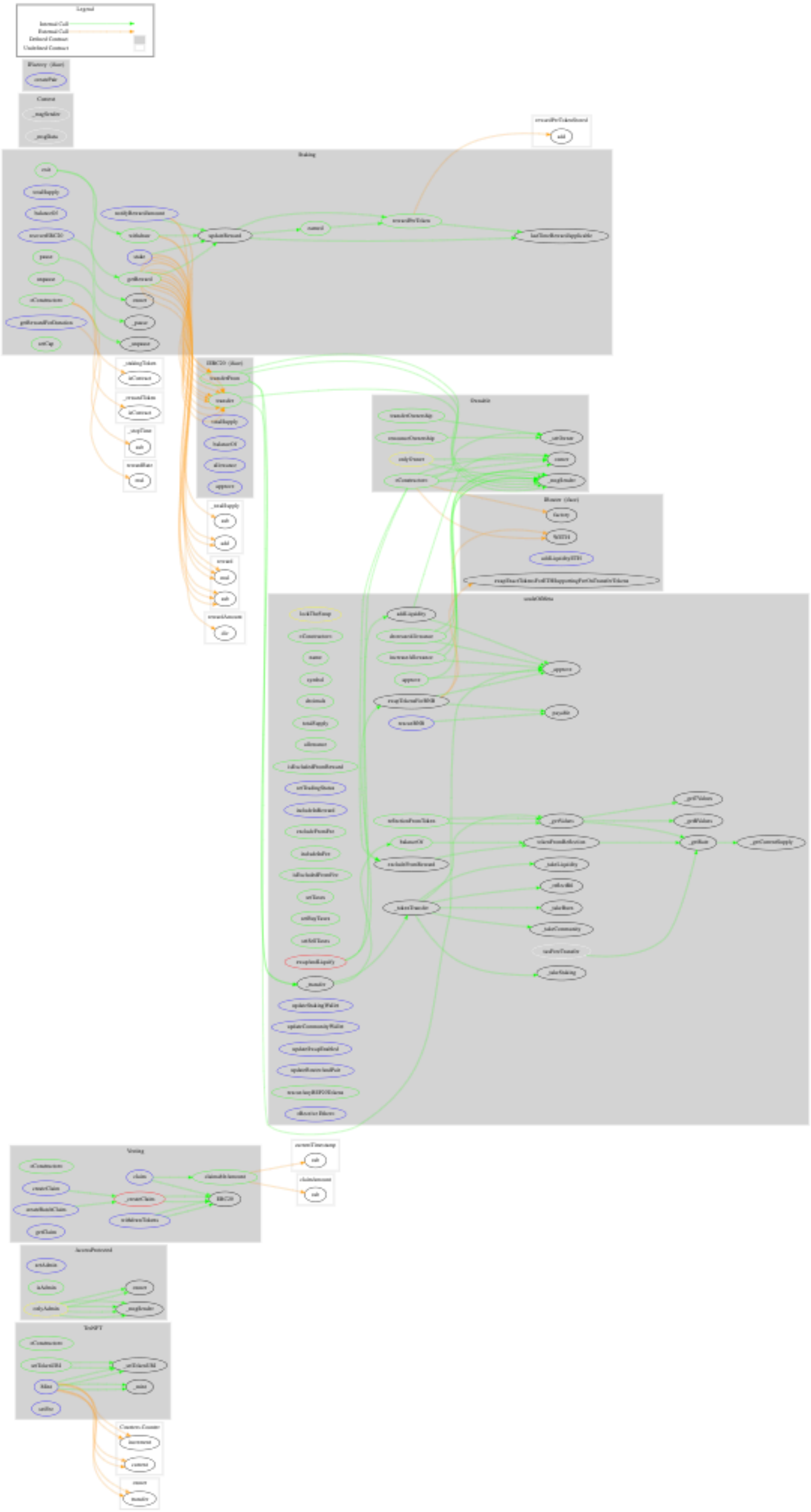
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/ Create/ Create2
1.0	yes					

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)

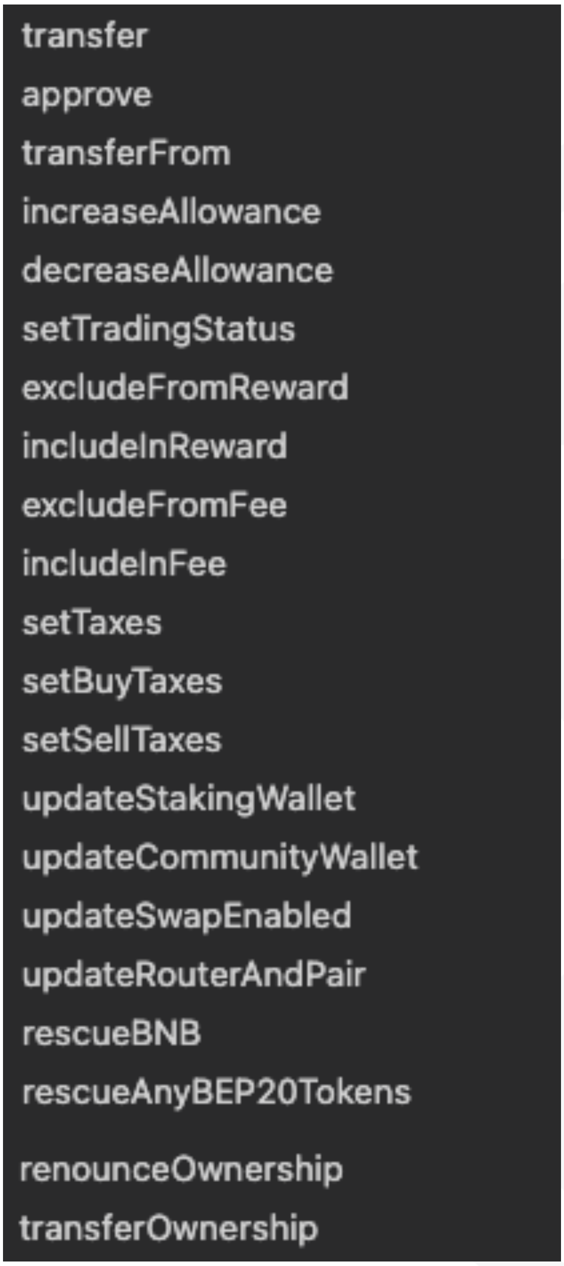
Correct implementation of Token standard

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	✓	✓	✓

ERC721				
Function	**Description**	**Exist**	**Tested**	**Verified**
BalanceOf	Count all NFTs assigned to an owner	✓	✓	✓
OwnerOf	Find the owner of an NFT	✓	✓	✓
SafeTransferFrom	Transfers the ownership of an NFT from one address to another address	✓	✓	✓
SafeTransferFrom	See above - Difference is that this function has an extra data parameter	✓	✓	✓
TransferFrom	Transfer ownership of an NFT	✓	✓	✓
Approve	Change or reaffirm the approved address for an NFT	✓	✓	✓
SetApprovalForAll	Enable or disable approval for a third party ("operator") to manage all of `msg.sender`'s assets	✓	✓	✓
GetApproved	Get the approved address for a single NFT	✓	✓	✓
IsApprovedForAll	Query if an address is an authorized operator for another address	✓	✓	✓
SupportsInterface	Query if a contract implements an interface	✓	✓	✓
Name	Provides information about the name	✓	✓	✓
Symbol	Provides information about the symbol	✓	✓	✓
TokenURI	Provides information about the TokenUri	✓	✓	✓

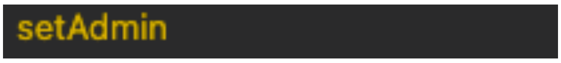
Write functions of contract v1.0

Token



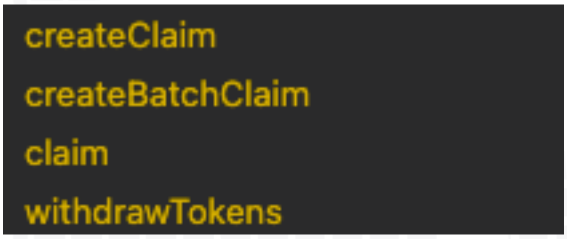
transfer
approve
transferFrom
increaseAllowance
decreaseAllowance
setTradingStatus
excludeFromReward
includeInReward
excludeFromFee
includeInFee
setTaxes
setBuyTaxes
setSellTaxes
updateStakingWallet
updateCommunityWallet
updateSwapEnabled
updateRouterAndPair
rescueBNB
rescueAnyBEP20Tokens
renounceOwnership
transferOwnership

AccessProtected



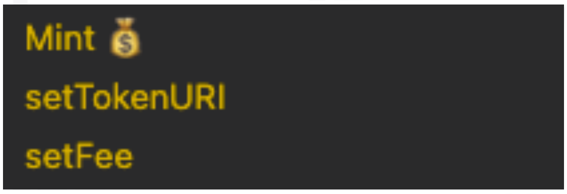
setAdmin

Vesting



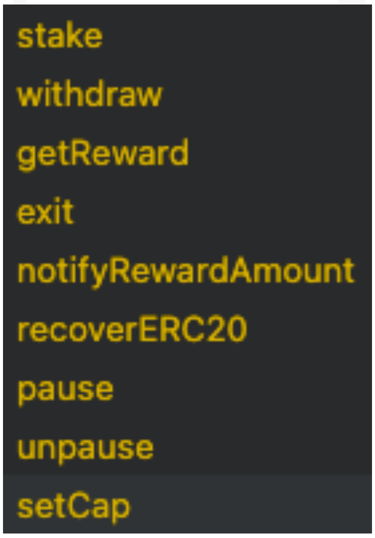
createClaim
createBatchClaim
claim
withdrawTokens

NFT Minter



Mint 💰
setTokenURI
setFee

Staking



stake
withdraw
getReward
exit
notifyRewardAmount
recoverERC20
pause
unpause
setCap

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	3000000000		

Comments:

v1.0

- Everybody can mint new nft

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Tokens will be burned while tx

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	✓	✓	✗

Comments:

v1.0

- Owner can pause contract

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

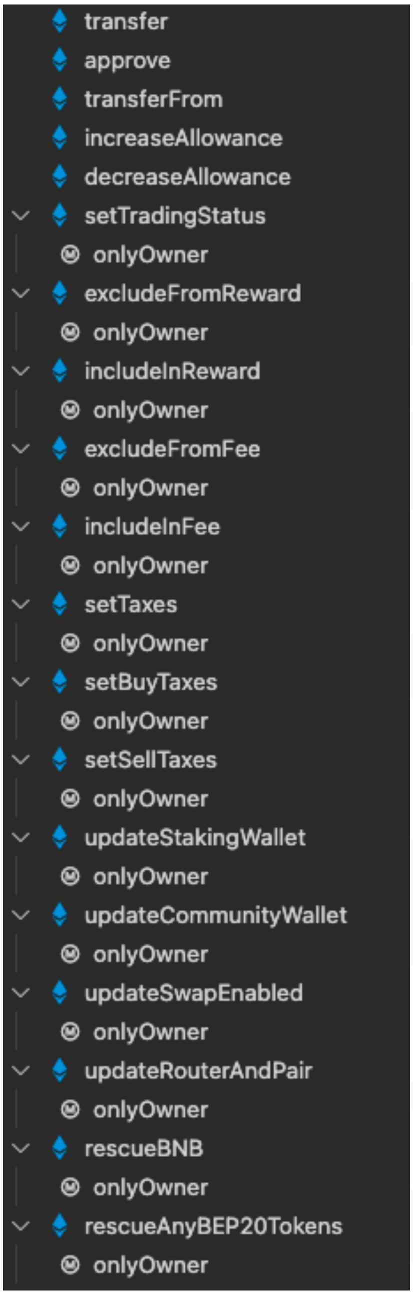
Legend

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

Modifiers and public functions

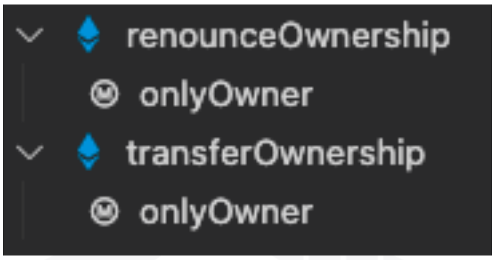
v1.0

Token



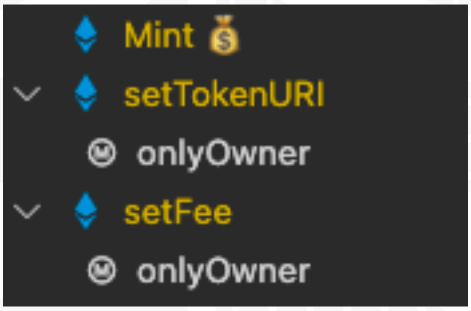
```
transfer
approve
transferFrom
increaseAllowance
decreaseAllowance
setTradingStatus
  onlyOwner
excludeFromReward
  onlyOwner
includeInReward
  onlyOwner
excludeFromFee
  onlyOwner
includeInFee
  onlyOwner
setTaxes
  onlyOwner
setBuyTaxes
  onlyOwner
setSellTaxes
  onlyOwner
updateStakingWallet
  onlyOwner
updateCommunityWallet
  onlyOwner
updateSwapEnabled
  onlyOwner
updateRouterAndPair
  onlyOwner
rescueBNB
  onlyOwner
rescueAnyBEP20Tokens
  onlyOwner
```

Ownable



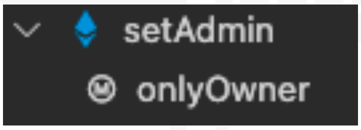
```
renounceOwnership
  onlyOwner
transferOwnership
  onlyOwner
```

NFT Minter



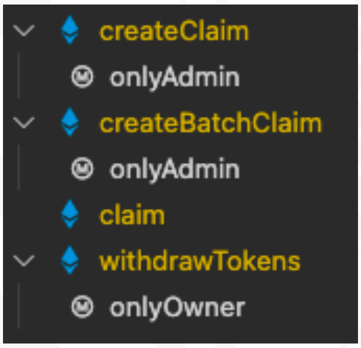
```
Mint 💰
setTokenURI
  onlyOwner
setFee
  onlyOwner
```

AccessProtocol



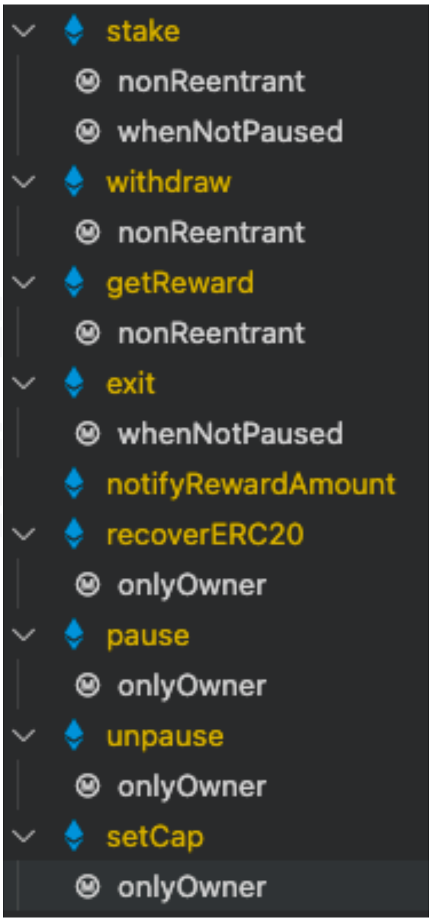
```
setAdmin
  onlyOwner
```

Vesting



```
createClaim
  onlyAdmin
createBatchClaim
  onlyAdmin
claim
withdrawTokens
  onlyOwner
```

Staking



```
stake
  nonReentrant
  whenNotPaused
withdraw
  nonReentrant
getReward
  nonReentrant
exit
  whenNotPaused
notifyRewardAmount
recoverERC20
  onlyOwner
pause
  onlyOwner
unpause
  onlyOwner
setCap
  onlyOwner
```

Note: Not listed functions are functions from library

Comments

- Deployer can set following state variables without any limitations
  - NFT Minter
    - fee

- Deployer can enable/disable following state variables
  - Token

- swapEnabled
- _isExcludedFromFee
- _isExcluded
- _excluded
- tradingEnabled
- swapEnabled

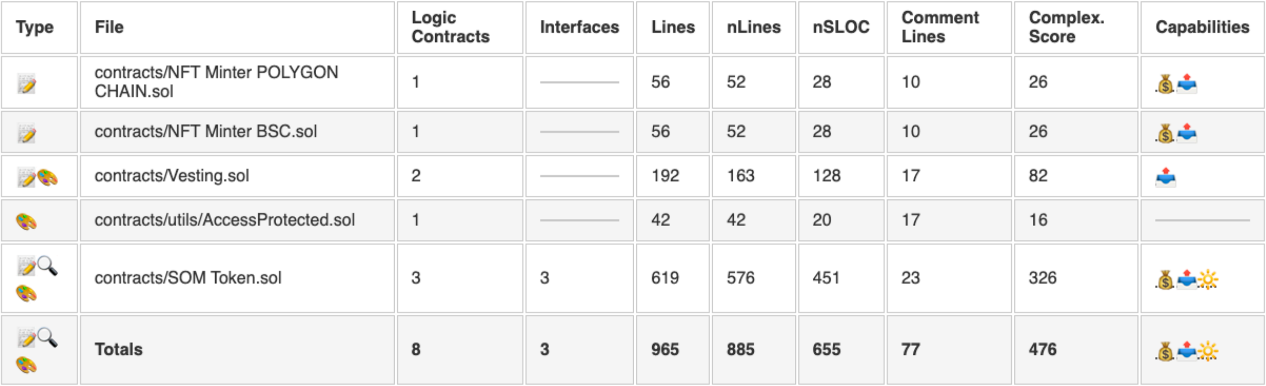
- Deployer can set following addresses
    - Token
        - router
        - pair
        - communityAddress
        - stakingAddress
    - NFT Minter
        - _tokenURIs[tokenId]
    - AccessProtected
        - _admins[admin]
- Vesting
    - Only admin can create new claim
        - If “inUnlockedAmount” of a claim is 0 the calculation in L160 will return 0
            - L160: (_claim.inUnlockedAmount*_claim.totalAmount)/100
    - Claim function
        - We recommend to check for “unclaimedAmount >0” in L180
        - Set state variable before transferring

- Following variables are not used in the contract or has no functionality
    - Token
        - swapEnabled
        - tradingEnabled
        - swapping
        - userLastSell
        - UserLastSell struct
- Minter
    - Mint function
        - Sent value will be transferred to owner. In this case the owner will be the zero address because owner is not set in the Mint function. The funds are lost
- Staking
    - If the stakingCap is set to totalSupply you are not able to stake. Investors have to wait for that someone withdraw to reduce the totalsupply

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/NFT Minter POLYGON CHAIN.sol	1	—	56	52	28	10	26	💰📥
📝	contracts/NFT Minter BSC.sol	1	—	56	52	28	10	26	💰📥
📝🎨	contracts/Vesting.sol	2	—	192	163	128	17	82	📥
🎨	contracts/utils/AccessProtected.sol	1	—	42	42	20	17	16	—
📝🔍🎨	contracts/SOM Token.sol	3	3	619	576	451	23	326	💰📥☀️
📝🔍🎨	**Totals**	**8**	**3**	**965**	**885**	**655**	**77**	**476**	💰📥☀️

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

Issue	File	Type	Line	Description
#1	Staking	recoverERC20	4	Owner can withdraw contract balance of Staking token.

Low issues

Issue	File	Type	Line	Description
#1	Minters	A floating pragma is set	4	The current pragma Solidity directive is „"^0.8.4"".
#2	Token	A floating pragma is set	3	The current pragma Solidity directive is „"^0.8.4"".
#3	Vesting	Missing Zero Address Validation (missing-zero-check)	76	Check that the address is not zero
#4	Token	Missing Zero Address Validation (missing-zero-check)	182, 582	Check that the address is not zero
#5	Minters	Local variables shadowing	32	Rename the local variables that shadow another component

#6	Token	Local variables shadowing	459, 228	Rename the local variables that shadow another component
#7	Minters	Missing Events Arithmetic	52	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	Token	State variables that could be declared constant (constable-states)	116	Add the `constant` attributes to state variables that never change
#2	Token	Functions that are not used	533, 523, 548, 588	Remove unused functions
#3	Main	Misspelling	See description	Change following words: - Make sure to change it everywhere else as well.
#4	All	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.
#5	Token	Wrong SPDX License	2	NOLICENSE is wrong, please choose one of the Identifier on this page https://spdx.org/licenses/

Audit Comments

15. April 2022:

- Read whole report for more information. Please read “modifiers and public functions” section carefully

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	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