SCRL

Preliminary Audit Report

MoonBag Security Assessment

MoonBag Security Assessment

PRELIMINARY AUDIT REPORT

Security Assessment by SCRL on **Thursday, May 2, 2024**

SCRL is deliver a security solution for Web3 projects by expert security researchers.

Executive Summary

For this security assessment, SCRL received a request on Tuesday, April 30, 2024

Client	Language	Audit Method	Confidential	Network Chain	Contract
MoonBag	**Solidity**	**Whitebox**	**Public**	**Ethereum**	**0xa7F4195F10F1a62B102bD683eAB131d657A6c6e4**

Report Version	Twitter	Telegram	Website
1.0	-	-	-

Scoring:

Vulnerability Summary

6	6	0	0	0	0
Total Findings	Unresolved	Resolved	Mitigate	Acknowledge	Decline

▪	0 **Critical**		Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
▪	0 **High**		High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.
▪	1 **Medium**	1 Unresolved	It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
▪	2 **Low**	2 Unresolved	While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.
▪	0 **Very Low**		Very Low severity is used for minor security concerns that have minimal impact and are generally of low risk.
▪	2 **Informational**	2 Unresolved	Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide additional information, recommendations
▪	1 **Gas-optimization**	1 Unresolved	Suggestions for more efficient algorithms or improvements in gas usage, even if the current code is already secure.

Audit Scope:

File	SHA-1 Hash
src/MoonBag.sol	7c52e9a45686a18ba55839ae97fca05d3006e64b

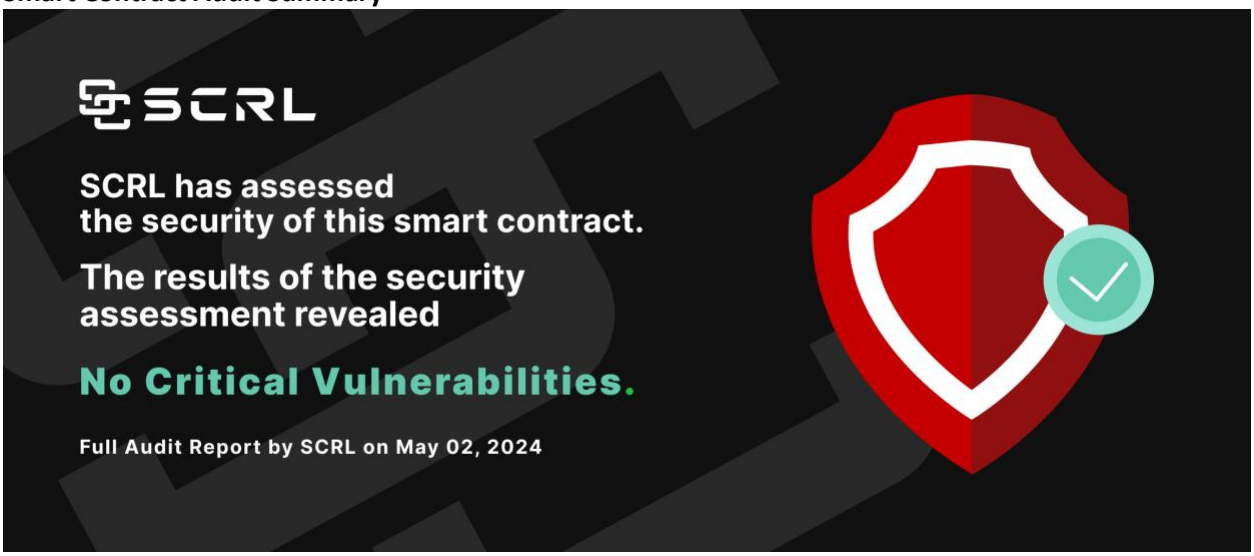
Audit Version History:

Version	Date	Description
1.0	Thursday, May 2, 2024	Preliminary Report

Audit information:

Request Date	Audit Date	Re-assessment Date
Tuesday, April 30, 2024	Thursday, May 2, 2024	-

Smart Contract Audit Summary



Security Assessment Author

Auditor:	**Mark K.**	[Security Researcher | Redteam]
	Kevin N.	[Security Researcher | Web3 Dev]
	Yusheng T.	[Security Researcher | Incident Response]
Document Approval:	**Ronny C.**	CTO & Head of Security Researcher
	Chinnakit J.	CEO & Founder

Digital Sign

Disclaimer

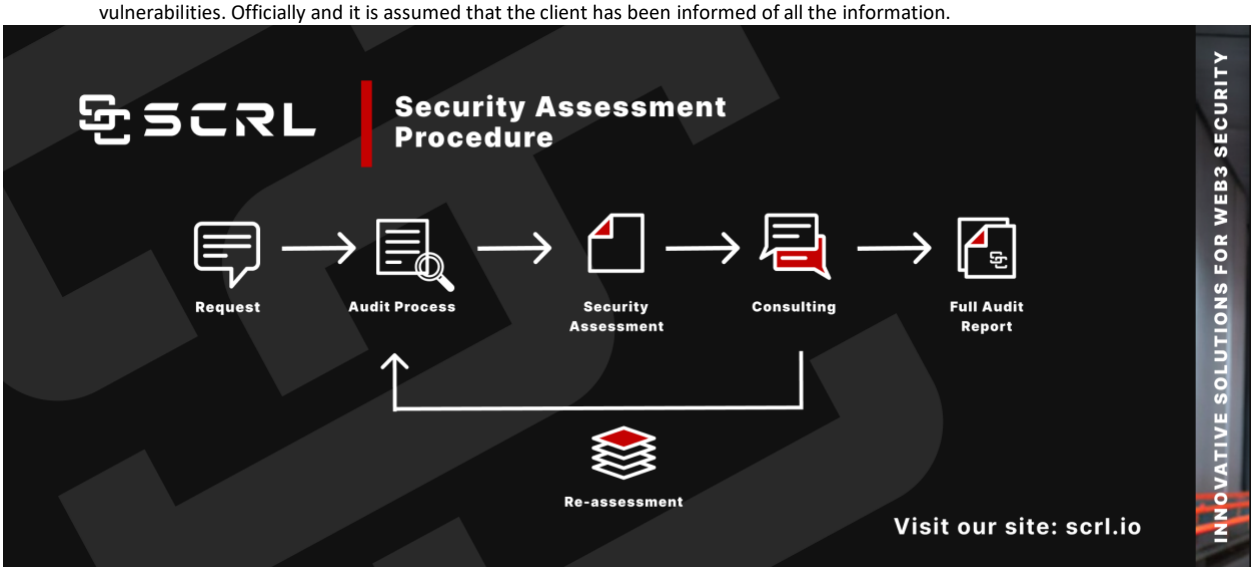
Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as "**Source code**".
And **SCRL** hereinafter referred to as "**Service Provider**", the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as "**Service User**" and the
Service User agrees not to be held liable to the **service provider** in any case. By contract
Service Provider to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.
If **the service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

Security Assessment Is Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.

SCRL disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull, Exploit, Exit Scam.

Security Assessment Procedure

1. **Request** The client must submit a formal request and follow the procedure. By submitting the source code and agreeing to the terms of service.
2. **Audit Process** Check for vulnerabilities and vulnerabilities from source code obtained by experts using formal verification methods, including using powerful tools such as Static Analysis, SWC Registry, Dynamic Security Analysis, Automated Security Tools, CWE, Syntax & Parameter Check with AI ,WAS (Warning Avoidance System a python script tools powered by SCRL).
3. **Security Assessment** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
4. **Consulting** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
    a. **Re-assessment** Reassess the security when the client implements the source code improvements and if the client is satisfied with the results of the audit. We will proceed to the next step.
5. **Full Audit Report** SCRL provides clients with official security assessment reports informing them of risks and vulnerabilities. Officially and it is assumed that the client has been informed of all the information.



Risk Rating

Risk rating using this commonly defined: $Risk\ rating\ =\ impact\ *\ confidence$

Impact The severity and potential impact of an attacker attack
Confidence Ensuring that attackers expose and use this vulnerability

Confidence / Impact [Likelihood]	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,
$Risk\ rating\ =\ impact\ *\ confidence$
It is categorized into
7 categories severity based

Gas-optimization | Informational | Very Low | Low | Medium | High | Critical

For **Informational** & **Non-class/Optimization/Best-practices will** not be counted as **severity**

Category

Centralization
Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission

Economics Risk
Economics Risk is Risks that may affect the economic mechanism system, such as the ability to increase Mint token

Logical Issue
Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.

Authorization
Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.

Mathematical
Mathematical Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.

Naming Conventions
Naming Conventions naming variables that may affect code understanding or naming inconsistencies

Security Risk
Security Risk of loss or damage if it's no mitigate

Coding Style
Coding Style is Tips coding for efficiency performance

Best Practices
Best Practices is suggestions for improvement

Optimization
Optimization is performance improvement

Gas Optimization
Gas Optimization is increase performance to avoid expensive gas

Dead Code
Dead Code having unused code This may result in wasted resources and gas fees.

Table Of Content

Source Units in Scope

Source Units Analyzed: 1
Source Units in Scope: 1 (100%)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	src/MoonBag.sol	1		196	180	120	26	83	Σ
📝	Totals	1		196	180	120	26	83	Σ

Legend: [—]

- **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, ...)

Visibility, Mutability, Modifier function testing

Components

Contracts	Libraries	Interfaces	Abstract
1	0	0	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
12	0

External	Internal	Private	Pure	View
1	17	0	0	6

StateVariables

Total	Public
6	0

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.18				

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

TryCatch	Σ Unchecked
	yes

Dependencies / External Imports

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
CEN-01	Centralization Risk	Medium	Centralization	-
SEC-01	Local variables shadowing (shadowing-local)	Low	Naming Conventions	-
SEC-02	Mark public functions as external where possible	Low	Best Practices	-
SEC-03	Conformity to Solidity naming conventions (naming-convention)	Informational	Naming Conventions	-
SEC-04	Functions that are not used (dead-code)	Informational	Dead Code	-
GAS-01	Use Custom Errors	Gas-optimization	Gas Optimization	-

CEN-01: Centralization Risk

Vulnerability Detail	Severity	Location	Category	Status
Centralization Risk	Medium	Check on finding	Centralization	-

Finding:

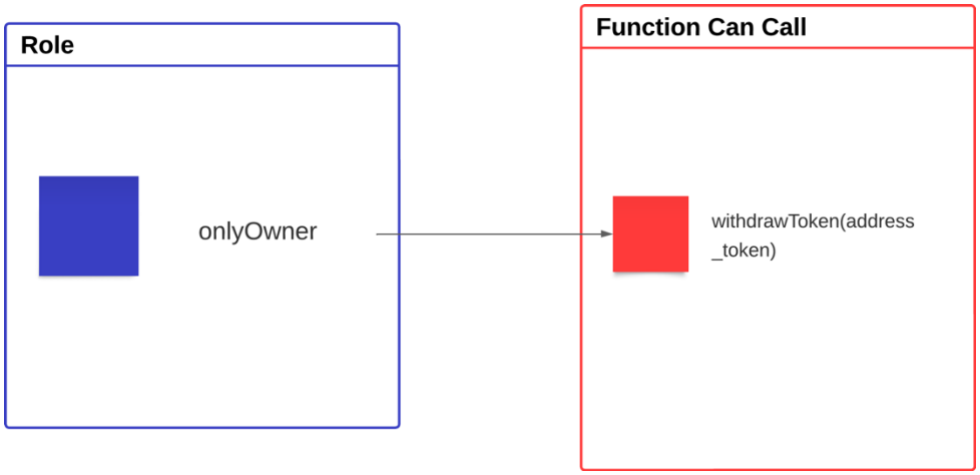
```
function withdrawToken(address _token) external onlyOwner {
```

Explain Function Capability:

The contract provides several functions:

1. function withdrawToken(address _token):
   The withdrawToken function allows the contract owner to withdraw any ERC20 tokens held by the contract.

Centralization Risk



Recommendation:
In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

Alleviation:
-

SEC-01: Local variables shadowing (shadowing-local)

Vulnerability Detail	Severity	Location	Category	Status
Local variables shadowing (shadowing-local)	Low	Check on finding	Naming Conventions	-

Finding:

❌ MoonBag._approve(address,address,uint256).owner (src/MoonBag.sol:166) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

❌ MoonBag._spendAllowance(address,address,uint256).owner (src/MoonBag.sol:178) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

❌ MoonBag.allowance(address,address).owner (src/MoonBag.sol:81) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

❌ MoonBag.approve(address,uint256).owner (src/MoonBag.sol:87) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

❌ MoonBag.decreaseAllowance(address,uint256).owner (src/MoonBag.sol:110) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

❌ MoonBag.increaseAllowance(address,uint256).owner (src/MoonBag.sol:104) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

❌ MoonBag.transfer(address,uint256).owner (src/MoonBag.sol:73) shadows:
- Ownable.owner() (@openzeppelin/contracts/access/Ownable.sol#43-45) (function)

Recommendation:
Rename the local variables that shadow another component.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Alleviation:
-

SEC-02: Mark public functions as external where possible

Vulnerability Detail	Severity	Location	Category	Status
Mark public functions as external where possible, to enhance contract's control-flow readability	Low	Check on finding	Best Practices	-

Finding:

❌ The following public functions could be turned into external in MoonBag (src/MoonBag.sol:8-196) contract:

Recommendation:

Mark public functions as external where it is possible

Reference: https://github.com/pessimistic-io/slitherin/blob/master/docs/public_vs_external.md

Alleviation:

-

SEC-03: Conformity to Solidity naming conventions

Vulnerability Detail	Severity	Location	Category	Status
Conformity to Solidity naming conventions (naming-convention)	Informational	Check on finding	Naming Conventions	-

Finding:

❌ Parameter MoonBag.withdrawToken(address)._token (src/MoonBag.sol:191) is not in mixedCase

Recommendation:

Follow the Solidity [naming convention] (https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions).

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Alleviation:

-

SEC-04: Functions that are not used (dead-code)

Vulnerability Detail	Severity	Location	Category	Status
Functions that are not used (dead-code)	Informational	Check on finding	Dead Code	-

Finding:

❌ MoonBag._burn(address,uint256) (src/MoonBag.sol:149-163) is never used and should be removed

Recommendation:

Remove unused functions.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Alleviation:

-

GAS-01: Use Custom Errors

Vulnerability Detail	Severity	Location	Category	Status
Use Custom Errors	-	Check on finding	Gas Optimization	-

Finding:

File: MoonBag.sol

```
112:        require(currentAllowance >= subtractedValue, "ERC20:allowance<0");

125:        require(from != address(0), "ERC20:From 0");

126:        require(to != address(0), "ERC20:To 0");

128:        require(fromBalance >= amount, "ERC20:amount>balance");

139:        require(account != address(0), "ERC20:address(0)");

150:        require(account != address(0), "ERC20:address(0)");

154:        require(accountBalance >= amount, "ERC20:amount>balance");

170:        require(owner != address(0), "ERC20:FromAddress(0)");

171:        require(spender != address(0), "ERC20:ToAddress(0)");

184:        require(currentAllowance >= amount, "ERC20: insufficient allowance");
```

Recommendation:

Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.
https://blog.soliditylang.org/2021/04/21/custom-errors/

Alleviation:

-

SWC Findings

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Complete	No risk
SWC-101	Integer Overflow and Underflow	Complete	No risk
SWC-102	Outdated Compiler Version	Complete	No risk
SWC-103	Floating Pragma	Complete	No risk
SWC-104	Unchecked Call Return Value	Complete	No risk
SWC-105	Unprotected Ether Withdrawal	Complete	No risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Complete	No risk
SWC-107	Reentrancy	Complete	No risk
SWC-108	State Variable Default Visibility	Complete	No risk
SWC-109	Uninitialized Storage Pointer	Complete	No risk
SWC-110	Assert Violation	Complete	No risk
SWC-111	Use of Deprecated Solidity Functions	Complete	No risk
SWC-112	Delegatecall to Untrusted Callee	Complete	No risk
SWC-113	DoS with Failed Call	Complete	No risk
SWC-114	Transaction Order Dependence	Complete	No risk
SWC-115	Authorization through tx.origin	Complete	No risk

SWC-116	Block values as a proxy for time	Complete	No risk
SWC-117	Signature Malleability	Complete	No risk
SWC-118	Incorrect Constructor Name	Complete	No risk
SWC-119	Shadowing State Variables	Complete	No risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Complete	No risk
SWC-121	Missing Protection against Signature Replay Attacks	Complete	No risk
SWC-122	Lack of Proper Signature Verification	Complete	No risk
SWC-123	Requirement Violation	Complete	No risk
SWC-124	Write to Arbitrary Storage Location	Complete	No risk
SWC-125	Incorrect Inheritance Order	Complete	No risk
SWC-126	Insufficient Gas Griefing	Complete	No risk
SWC-127	Arbitrary Jump with Function Type Variable	Complete	No risk
SWC-128	DoS With Block Gas Limit	Complete	No risk
SWC-129	Typographical Error	Complete	No risk
SWC-130	Right-To-Left-Override control character (U+202E)	Complete	No risk
SWC-131	Presence of unused variables	Complete	No risk
SWC-132	Unexpected Ether balance	Complete	No risk

SWC-133	Hash Collisions With Multiple Variable Length Arguments	Complete	No risk
SWC-134	Message call with hardcoded gas amount	Complete	No risk
SWC-135	Code With No Effects	Complete	No risk
SWC-136	Unencrypted Private Data On-Chain	Complete	No risk

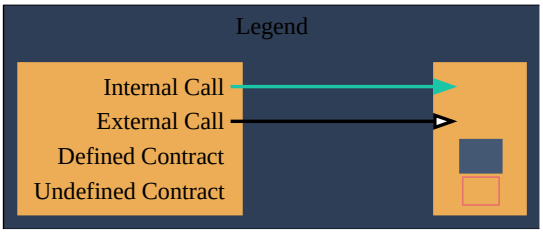
Contracts Description Table

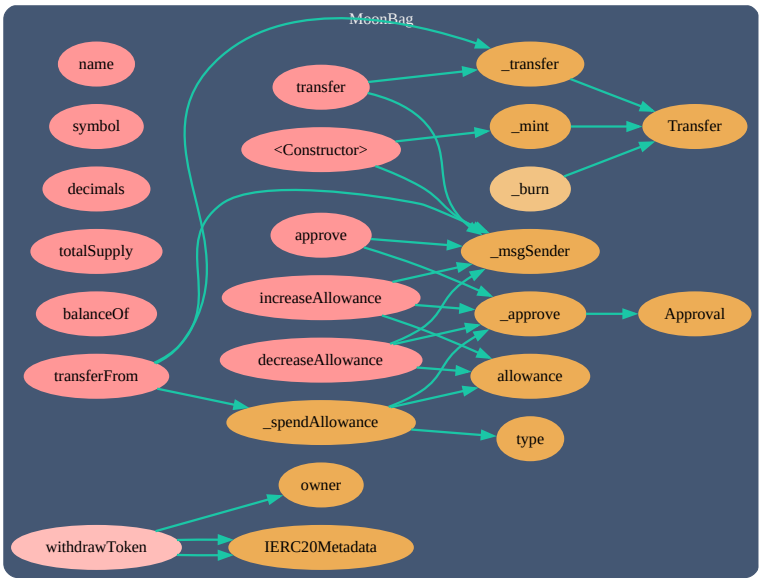
Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
MoonBag	Implementation	IERC20Metadata, Ownable		
└		Public ❗	🛑	NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_spendAllowance	Internal 🔒	🛑	
└	withdrawToken	External ❗	🛑	onlyOwner

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Call Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract


MoonBag
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
<Constructor>
approve
increaseAllowance
decreaseAllowance
_spendAllowance
_transfer
_mint
_burn
_msgSender
_approve
allowance
type
Transfer
Approval
owner
withdrawToken
IERC20Metadata

UML Class Diagram

MoonBag
MoonBag.sol

Private:
- _balances: mapping(address=>uint256)
- _allowances: mapping(address=>mapping(address=>uint256))
- _totalSupply: uint256
- _name: string
- _symbol: string
- _decimals: uint8

Internal:
- _transfer(from: address, to: address, amount: uint256)
- _mint(account: address, amount: uint256)
- _burn(account: address, amount: uint256)
- _approve(owner: address, spender: address, amount: uint256)
- _spendAllowance(owner: address, spender: address, amount: uint256)

External:
- withdrawToken(_token: address) <<onlyOwner>>

Public:
- constructor(__name: string, __symbol: string, __totalSupply: uint256, __decimals: uint8)
- name(): string
- symbol(): string
- decimals(): uint8
- totalSupply(): uint256
- balanceOf(account: address): uint256
- transfer(to: address, amount: uint256): bool
- allowance(owner: address, spender: address): uint256
- approve(spender: address, amount: uint256): bool
- transferFrom(from: address, to: address, amount: uint256): bool
- increaseAllowance(spender: address, addedValue: uint256): bool
- decreaseAllowance(spender: address, subtractedValue: uint256): bool

About SCRL

SCRL (Previously name SECURI LAB) was established in 2020, and its goal is to deliver a security solution for Web3 projects by expert security researchers. To verify the security of smart contracts, they have developed internal tools and KYC solutions for Web3 projects using industry-standard technology. SCRL was created to solve security problems for Web3 projects. They focus on technology for conciseness in security auditing. They have developed Python-based tools for their internal use called WAS and SCRL. Their goal is to drive the crypto industry in Thailand to grow with security protection technology.

Follow Us On:

Website	https://scrl.io/
Twitter	https://twitter.com/scrl_io
Telegram	https://t.me/scrl_io
Medium	https://scrl.medium.com/