Security Audit

Full Audit Report For MoonBull

Client: MoonBull
https://www.moonbull.io/

September 9, 2025

Version: 1.1

Table of Contents

1 Disclaimer

This security assessment does not guarantee the security of the program instruction received from the client, herein referred to as “Source code.”

The Service Provider will not be held liable for any legal liability arising from errors in the security assessment. The responsibility for ensuring the security of the program instruction will be solely with the Client, herein referred to as “Service User.” The Service User agrees not to hold the Service Provider liable for any such liability.

By contract, the Service Provider is committed to conducting security assessments with integrity, professional ethics, and transparency in delivering security assessments to users. The Service Provider reserves the right to postpone the delivery of the security assessment if necessary, regardless of the reason for the delay. The Service Provider will not be held responsible for any delayed security assessments.

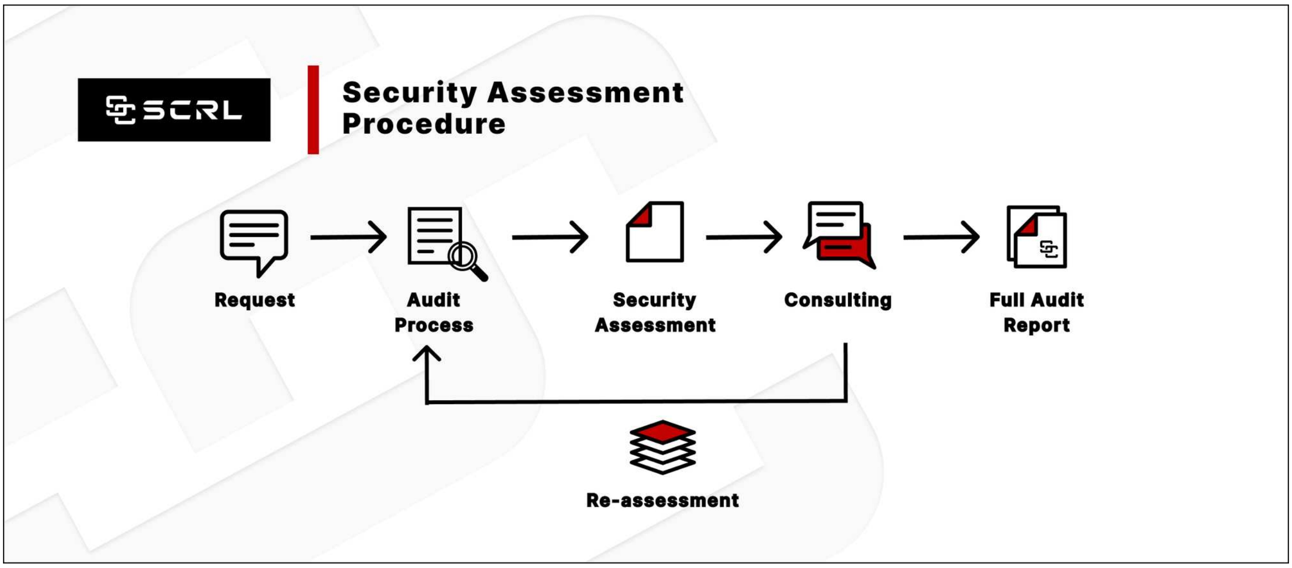
If the Service Provider identifies a vulnerability, we will notify the Service User via a Preliminary Report, which will be maintained in confidence for security purposes. The Service Provider disclaims responsibility in the event of any attacks occurring before or after conducting a security assessment. All responsibility for ensuring the security of the program instruction will be solely with the Service User.

Please be advised that the Security Assessment is not an investment advisory document and should not be interpreted as such. We hereby disclaim any liability arising from any investment:

- 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.**
- **Cryptocurrency and digital token involve high risks; investors may lose all investment money and should study information carefully and make investments according to own risk profile.**
- Please thoroughly comprehend any aspect of a project, including liquidity checks and pre-sale stages, and assume the associated risks independently. We hereby disclaim any liability for any investment decisions made by you.

For full Legal / TOS / Privacy Policy please visit: *https://scrl.io/legal*

2 Security Assessment Procedure



- **Request:** The client must submit a formal request and follow the procedure. By submitting the source code and agreeing to the terms of service.
- **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).
- **Security Assessment:** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
- **Consulting:** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
- **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.
- **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.

3 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	**Informational**	**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

Severity Risk	Description
Critical	Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
High	High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.
Medium	It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
Low	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.
Informational	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

4 Category

Category	Description
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	Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.
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.
Input Validation	Proper input validation is essential to ensure that a smart contract processes only valid and anticipated data. Failure to implement robust input validation mechanisms may lead to various security vulnerabilities, including logic manipulation, unauthorized access, and unintended contract behavior.

5 Executive Summary

For this security assessment, SCRL received a request on **August 6, 2025**

Audit Score	Project Website	X (Twitter)	Telegram
9.7	https://www.moonbull.io/	https://x.com/MoonBullX	https://t.me/MoonBullCoin

Client	Confidential	Audit Method	Language
MoonBull	Public	Static Analysis + Dynamic Analysis + Manual Review + Formal Verification + Hyper Fuzzing + WAS (Custom Detector)	Solidity

Vulnerabilities Summary:

11	9	0	2	0
Total Findings	**Resolved**	**Mitigated**	**Acknowledge**	**Declined**

Count	Severity	Status	Description
0	Critical		Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
3	High	Resolved: 3	High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users funds and data.
0	Medium		It essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
1	Low	Acknowledge: 1	While low-severity issues can be less urgent, it still advisable to address them to improve the overall security posture of the smart contract.
7	Informational	Resolved: 6 Acknowledge: 1	Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide information

6 Audit Information

Audit Scope:

File	SHA1-Hash
src/MoonBull.sol	a7b3c9e11185343cf5730d34a999f4d6fa57ba71

Onchain Scope:

Chain	Contract Address
Ethereum	0x0E909a02FaC3D016E88AE9e0B2991645Cd07f0d4

Security Assessment Author:

Name	Role
Chinnakit J.	CEO & Founder
Ronny C.	CTO & Head of Security Researcher
Mark K.	Security Researcher

Smart Contract Audit Summary:

Digital Sign:

ID: 2F3C5EB9-EFA6-4DBF-89D0-DDCE46DA363B
Digitally signed by <contact@scrl.io>
September 09, 2025 10:24 PM +07

SWC Checklist

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Completed	No Risk
SWC-101	Integer Overflow and Underflow	Completed	No Risk
SWC-102	Outdated Compiler Version	Completed	No Risk
SWC-103	Floating Pragma	Completed	No Risk
SWC-104	Unchecked Call Return Value	Completed	No Risk
SWC-105	Unprotected Ether Withdrawal	Completed	No Risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Completed	No Risk
SWC-107	Reentrancy	Completed	No Risk
SWC-108	State Variable Default Visibility	Completed	No Risk
SWC-109	Uninitialized Storage Pointer	Completed	No Risk
SWC-110	Assert Violation	Completed	No Risk
SWC-111	Use of Deprecated Solidity Functions	Completed	No Risk
SWC-112	Delegatecall to Untrusted Callee	Completed	No Risk
SWC-113	DoS with Failed Call	Completed	No Risk
SWC-114	Transaction Order Dependence	Completed	No Risk
SWC-115	Authorization through tx.origin	Completed	No Risk
SWC-116	Block values as a proxy for time	Completed	No Risk
SWC-117	Signature Malleability	Completed	No Risk
SWC-118	Incorrect Constructor Name	Completed	No Risk
SWC-119	Shadowing State Variables	Completed	No Risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Completed	No Risk
SWC-121	Missing Protection against Signature Replay Attacks	Completed	No Risk

ID	Title	Scanning	Result
SWC-122	Lack of Proper Signature Verification	Completed	No Risk
SWC-123	Requirement Violation	Completed	No Risk
SWC-124	Write to Arbitrary Storage Location	Completed	No Risk
SWC-125	Incorrect Inheritance Order	Completed	No Risk
SWC-126	Insufficient Gas Griefing	Completed	No Risk
SWC-127	Arbitrary Jump with Function Type Variable	Completed	No Risk
SWC-128	DoS With Block Gas Limit	Completed	No Risk
SWC-129	Typographical Error	Completed	No Risk
SWC-130	Right-To-Left-Override control character (U+202E)	Completed	No Risk
SWC-131	Presence of unused variables	Completed	No Risk
SWC-132	Unexpected Ether balance	Completed	No Risk
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Completed	No Risk
SWC-134	Message call with hardcoded gas amount	Completed	No Risk
SWC-135	Code With No Effects	Completed	No Risk
SWC-136	Unencrypted Private Data On-Chain	Completed	No Risk

OWASP Smart Contract Top 10 2025 Checklist

ID	Title	Scanning	Result
SC01:2025	Access Control Vulnerabilities	Completed	No Risk
SC02:2025	Price Oracle Manipulation	Completed	No Risk
SC03:2025	Logic Errors	Completed	No Risk
SC04:2025	Lack of Input Validation	Completed	No Risk
SC05:2025	Reentrancy Attacks	Completed	No Risk
SC06:2025	Unchecked External Calls	Completed	No Risk
SC07:2025	Flash Loan Attacks	Completed	No Risk
SC08:2025	Integer Overflow and Underflow	Completed	No Risk
SC09:2025	Insecure Randomness	Completed	No Risk
SC10:2025	Denial of Service (DoS) Attacks	Completed	No Risk

7 Findings List

0	3	0	1	7
Critical	High	Medium	Low	Informational

This security assessment report for **MoonBull** a total of **11 vulnerabilities.** The evaluation was conducted using the **Static Analysis + Dynamic Analysis + Manual Review + Formal Verification + Hyper Fuzzing + WAS (Custom Detector)** security assessment methodology.

ID	Vulnerabilities	Severity	Category	Status
BUG-01	Ownable Constructor Requirement	High	Bug	Resolved
BUG-02	Immutable Variables Limitation	High	Bug	Resolved
BUG-03	Stack Too Deep in _getValues()	High	Bug	Resolved
DEX-01	Router & Dex Pair Risk	Low	Best Practices	Acknowledge
NSM-01	Non-standard Mapping Name	Informational	Best Practices	Resolved
SNC-01	Conformity to Solidity naming conventions	Informational	Coding Style	Resolved
CEN-01	Centralization Risk	Informational	Centralization	Acknowledge
UBS-01	Using bools for storage incurs overhead	Informational	Gas Optimization	Resolved
PDM-01	Use of Post-Increment (i++) Instead of Pre-Increment (++i) in Loops	Informational	Gas Optimization	Resolved
DIV-01	Don't initialize variables with default value	Informational	Gas Optimization	Resolved
USR-01	Use shift Right/Left instead of division/multiplication if possible	Informational	Gas Optimization	Resolved

8 Findings

BUG-01: Ownable Constructor Requirement

Vulnerabilities	Severity	Category	Status
Ownable Constructor Requirement	High	Bug	Resolved

Description

The Ownable contract from OpenZeppelin requires the initialOwner to be passed to its constructor.

Recommendation

Refactor code:

```
constructor() Ownable(_msgSender()) {
    // initialization logic
}
```

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue by refactoringthe the code to fix this bug

Refactored:

```
    constructor() Ownable(_msgSender()) {
        _name = "MoonBull";
        _symbol = "$MOBU";
        _tTotal = 73_200_000_000 * 10**DECIMALS;
        _rTotal = (MAX - (MAX % _tTotal));

        _rOwned[_msgSender()] = _rTotal;

        // Exclude contract from rewards to prevent fee loops
        _isExcluded[address(this)] = 1;
        _excluded.push(address(this));

        // Exclude burn address from rewards
        _isExcluded[burnAddress] = 1;
        _excluded.push(burnAddress);

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

References

-

BUG-02: Immutable Variables Limitation

Vulnerabilities	Severity	Category	Status
Immutable Variables Limitation	High	Bug	Resolved

Description

Immutable variables _name and _symbol can only be assigned once—either at declaration or in the constructor.

Recommendation

Refactor code:

string private _name;

string private _symbol;

```
constructor() Ownable(_msgSender()) {
    _name = "MoonBull";
    _symbol = "$MOBU";
    // other initialization
}
```

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue by refactoringthe the code to fix this bug

Refactored:

```
contract MoonBull is Context, IERC20, IERC20Metadata, Ownable {
    string private _name;
    string private _symbol;

    }
```

References

-

BUG-03: Stack Too Deep in _getValues()

Vulnerabilities	Severity	Category	Status
Stack Too Deep in _getValues()	High	Bug	Resolved

Description

The function exceeds Solidity's limit of 16 local variables due to multiple return values and temporary variables.

Recommendation

Refactor return values into a struct to reduce stack usage.

Refactor code:

Example

```
function _tokenTransfer(address sender, address recipient, uint256 tAmount, bool takeFee) private {
        TransferValues memory values = _getValues(tAmount, takeFee);

        if (_isExcluded[sender]) {
            _tOwned[sender] = _tOwned[sender] - tAmount;
        }
        _rOwned[sender] = _rOwned[sender] - values.rAmount;

        if (_isExcluded[recipient]) {
            _tOwned[recipient] = _tOwned[recipient] + values.tTransferAmount;
        }
        _rOwned[recipient] = _rOwned[recipient] + values.rTransferAmount;

        if (takeFee) {
            _takeLiquidity(values.tLiquidity);
            _takeBurn(values.tBurn);
            _reflectFee(values.rFee, values.tFee);
        }

        emit Transfer(sender, recipient, values.tTransferAmount);
    }

    function _reflectFee(uint256 rFee, uint256 tFee) private {
        _rTotal = _rTotal - rFee;
        _tFeeTotal = _tFeeTotal + tFee;
        emit ReflectionDistributed(tFee);
    }

    struct TransferValues {
        uint256 rAmount;
        uint256 rTransferAmount;
        uint256 rFee;
        uint256 tTransferAmount;
        uint256 tFee;
        uint256 tLiquidity;
        uint256 tBurn;
    }


    function _getValues(uint256 tAmount, bool takeFee) private view returns (TransferValues memory) {
        (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity, uint256 tBurn) =
_getTValues(tAmount, takeFee);
        uint256 currentRate = _getRate();
```

```
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, tBurn, currentRate);

        return TransferValues(
            rAmount,
            rTransferAmount,
            rFee,
            tTransferAmount,
            tFee,
            tLiquidity,
            tBurn
        );
    }
```

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue by refactoringthe the code to fix this bug

References

-

DEX-01: Router & Dex Pair Risk

Vulnerabilities	Severity	Category	Status
Router & Dex Pair Risk	Low	Best Practices	Acknowledge

Description

The owner can call setRouter() and setDexPair() at any time. If set incorrectly, swapAndLiquify() will fail during a sell, causing transfer() to revert due to a failed swap or liquidity addition.

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.

References

-

NSM-01: Non-standard Mapping Name

Vulnerabilities	Severity	Category	Status
Non-standard Mapping Name	Informational	Best Practices	Resolved

Description

The mapping _launchCodes is used for token allowances but deviates from the standard _allowances, which may confuse auditors or tools relying on ERC-20 naming conventions.

Recommendation

Rename to _allowances for consistency.

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue

Refactored:

```
mapping(address => mapping(address => uint256)) private _allowances;
```

```
function allowance(address astronaut, address copilot) public view override returns (uint256) {
        return _allowances[astronaut][copilot];
    }
```

```
function increaseAllowance(address copilot, uint256 addedValue) public returns (bool) {
        address astronaut = _msgSender();
        _approve(astronaut, copilot, _allowances[astronaut][copilot] + addedValue);
        return true;
    }
```

```
function decreaseAllowance(address copilot, uint256 subtractedValue) public returns (bool) {
        address astronaut = _msgSender();
        uint256 currentAllowance = _allowances[astronaut][copilot];
        if (currentAllowance < subtractedValue) revert NotEnoughRocketFuel();
        unchecked {
            _approve(astronaut, copilot, currentAllowance - subtractedValue);
        }
        return true;
    }
```

```
function _approve(address astronaut, address copilot, uint256 amount) private {
        if (astronaut == address(0)) revert LaunchPadLocked();
        if (copilot == address(0)) revert LaunchPadLocked();

        _allowances[astronaut][copilot] = amount;
        emit Approval(astronaut, copilot, amount);
    }
```

```
function swapTokensForEth(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        if (_allowances[address(this)][address(uniswapV2Router)] < tokenAmount) {
            _approve(address(this), address(uniswapV2Router), type(uint256).max);
        }
```

```
        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }
```

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        if (_allowances[address(this)][address(uniswapV2Router)] < tokenAmount) {
            _approve(address(this), address(uniswapV2Router), type(uint256).max);
        }

        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0,
            0,
            burnAddress,
            block.timestamp
        );
    }
```

References

-

SNC-01: Conformity to Solidity naming conventions

Vulnerabilities	Severity	Category	Status
Conformity to Solidity naming conventions	Informational	Coding Style	Resolved

Finding Code

```
❌ Constant MoonBull._decimals (src/MoonBull.sol:48) is not in UPPER_CASE_WITH_UNDERSCORES
❌ Constant MoonBull.numTokensSellToAddToLiquidity (src/MoonBull.sol:71) is not in
UPPER_CASE_WITH_UNDERSCORES
❌ Function IUniswapV2Router02.WETH() (src/MoonBull.sol:25) is not in mixedCase
❌ Parameter MoonBull.calculateBurnFee(uint256)._amount (src/MoonBull.sol:423) is not in mixedCase
❌ Parameter MoonBull.calculateLiquidityFee(uint256)._amount (src/MoonBull.sol:419) is not in mixedCase
❌ Parameter MoonBull.calculateReflectionFee(uint256)._amount (src/MoonBull.sol:415) is not in
mixedCase
❌ Parameter MoonBull.setRouter(address)._router (src/MoonBull.sol:184) is not in mixedCase
```

Description

The smart contract does not fully adhere to Solidity's standard naming conventions. Consistent and conventional naming improves readability, maintainability, and helps avoid confusion or misinterpretation of a contract's intent.

Recommendation

It is recommended to refactor the contract to fully comply with Solidity's official naming conventions. This practice aligns with industry standards and facilitates more effective collaboration, auditing, and code comprehension.

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue

References

Solidity Style Guide – Solidity Docs *https://docs.soliditylang.org/en/latest/style-guide.html*

CEN-01: Centralization Risk

Vulnerabilities	Severity	Category	Status
Centralization Risk	Informational	Centralization	Acknowledge

Finding Code

```
45:      contract MoonBull is Context, IERC20, IERC20Metadata, Ownable {
92:      constructor() Ownable(_msgSender()) {
184:     function setRouter(address _router) external onlyOwner {
190:     function setDexPair(address pair, bool isPair) external onlyOwner {
206:     function renounceOwnership() public override onlyOwner {
```

Description

Centralization Risk

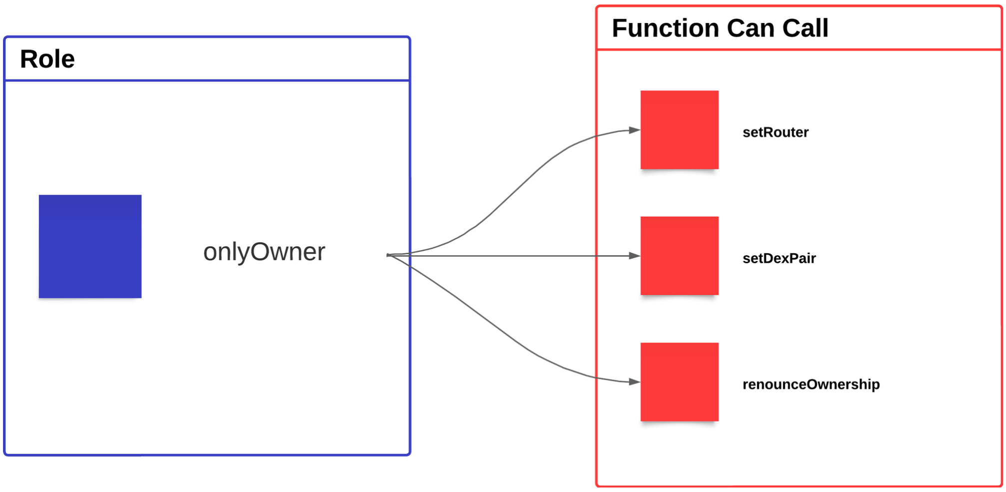


Figure 1 - This image shows who has privileges to call the function.

A Centralization Risk has been identified for the `Contract` entity, which has privileges set to `onlyOwner`. This restriction allows the execution of the following functions: `setRouter` , `setDexPair` , `renounceOwnership`

Each function possesses the following capabilities:

1. setRouter(address _router) - Updates the uniswapV2Router and emits a RouterUpdated event.
2. setDexPair(address pair, bool isPair) - Updates the isDexPair mapping and manages exclusion from rewards, emits a DexPairUpdated event.
3. renounceOwnership() - Transfers ownership to the zero address, effectively renouncing ownership.

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.

- Renounce the ownership and cannot turn back to claim a privileges to execution a function

OR

- Remove a contain centralization risk function and deployed a new contract

References

-

UBS-01: Using bools for storage incurs overhead

Vulnerabilities	Severity	Category	Status
Using bools for storage incurs overhead	Informational	Gas Optimization	Resolved

Finding Code

```
63:     mapping(address => bool) private _isExcluded;

64:     mapping(address => bool) public isDexPair;

69:     bool private inSwapAndLiquify;
```

Description

Using bool types in storage variables can lead to **unnecessary gas overhead** due to the way the Ethereum Virtual Machine (EVM) handles storage writes. Although a bool only requires 1 byte, the EVM stores data in 32-byte storage slots. When a bool is modified, the entire 32-byte storage slot must be read and rewritten due to the need to preserve surrounding data.

This behavior results in **extra gas consumption**, especially when multiple bool variables are packed into the same slot or modified frequently. Additionally, slot packing may complicate upgradeability and reduce predictability of gas costs.

Impact:

- Increased gas usage during storage writes
- Potential complications in upgradeable contracts or proxy patterns
- Reduced performance for large-scale contracts or those with frequent state changes

Recommendation

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past.

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue

References

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27

PDM-01: Use of Post-Increment (i++) Instead of Pre-Increment (++i) in Loops

Vulnerabilities	Severity	Category	Status
Use of Post-Increment (i++) Instead of Pre-Increment (++i) in Loops	Informational	Gas Optimization	Resolved

Finding Code

```
386:         for (uint256 i = 0; i < excludedLength; i++) {
```

Description

The contract uses the post-increment (i++) or post-decrement (i--) operator in a for-loop. While this is syntactically valid, using pre-increment (++i) or pre-decrement (--i) is marginally more gas-efficient in Solidity.

This is because post-increment creates a temporary copy of the variable before incrementing, whereas pre-increment directly increments the variable, avoiding unnecessary stack operations. Although the gas difference is small, this optimization is considered a best practice, especially within loops that may execute multiple iterations.

Recommendation

Replace all instances of `i++` / `i--` in loop expressions with `++i` / `--i` for slight gas savings and improved efficiency, especially when the return value of the increment/decrement expression is not used.

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue

References

-

DIV-01: Don't initialize variables with default value

Vulnerabilities	Severity	Category	Status
Don't initialize variables with default value	Informational	Gas Optimization	Resolved

Finding Code

```
386:        for (uint256 i = 0; i < excludedLength; i++) {
```

Description

In Solidity, variables are automatically assigned default values when declared. For example, uint and int default to 0, bool defaults to false, and address defaults to 0x000...000. Explicitly assigning these default values during variable declaration is **redundant** and results in **unnecessary bytecode and gas usage** during contract deployment.

Impact:

- Slight increase in contract size and deployment cost
- Reduced code clarity due to redundant assignments
- Missed opportunity for gas optimization

Recommendation

Avoid explicitly initializing variables to their default values. Rely on Solidity's automatic default initialization to reduce bytecode size and deployment gas costs.

Example *This is not fixed code*:

```
uint256 public count; // Automatically initialized to 0
```

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue

References

https://docs.soliditylang.org/en/latest/style-guide.html#initial-values

USR-01: Use shift Right/Left instead of division/multiplication if possible

Vulnerabilities	Severity	Category	Status
Use shift Right/Left instead of division/multiplication if possible	Informational	Gas Optimization	Resolved

Finding Code

```
255:        uint256 half = contractTokenBalance / 2;
```

Description

Solidity supports bitwise shift operations (<< and >>) which are **more gas-efficient** than using arithmetic multiplication or division when the operation involves powers of two. Using x * 2 or x / 2 can be replaced with x << 1 or x >> 1, respectively, for better performance and lower gas costs.

Recommendation

When multiplying or dividing by powers of two, replace the arithmetic operation with bitwise shift operators.

Example, This is not a fixed code

✅ **Preferred:**

```
uint256 result = value << 1; // equivalent to value * 2
uint256 result = value >> 2; // equivalent to value / 4
```

❌ **Avoid:**

```
uint256 result = value * 2;
uint256 result = value / 4;
```

Alleviation

[2025-08-08] MoonBull Team has already fixed this issue

References

-

A Appendix

A.1 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/MoonBull.sol	1	2	425	397	308	13	234	💰****Σ
📝🔍	**Totals**	**1**	**2**	**425**	**397**	**308**	**13**	**234**	💰****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, ...)

Document ID: 19810CD20B84A5ABDACBEEA9B95E6D1D

A.2 Visibility, Mutability, Modifier function testing

Components

📝****Contracts	📚****Libraries	🔍****Interfaces	🎨****Abstract
1	0	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
22	2

External	Internal	Private	Pure	View
8	22	17	3	15

StateVariables

Total	🌐****Public
20	6

Capabilities

Solidity Versions observed	🧪** Experimental Features**	💰** Can Receive Funds**	🖥** Uses Assembly**	💣** Has Destroyable Contracts**
`0.8.28`		`yes`	***	***

📤** 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/utils/Context.sol	1

A.3 Contracts Description Table

Contracts Description Table

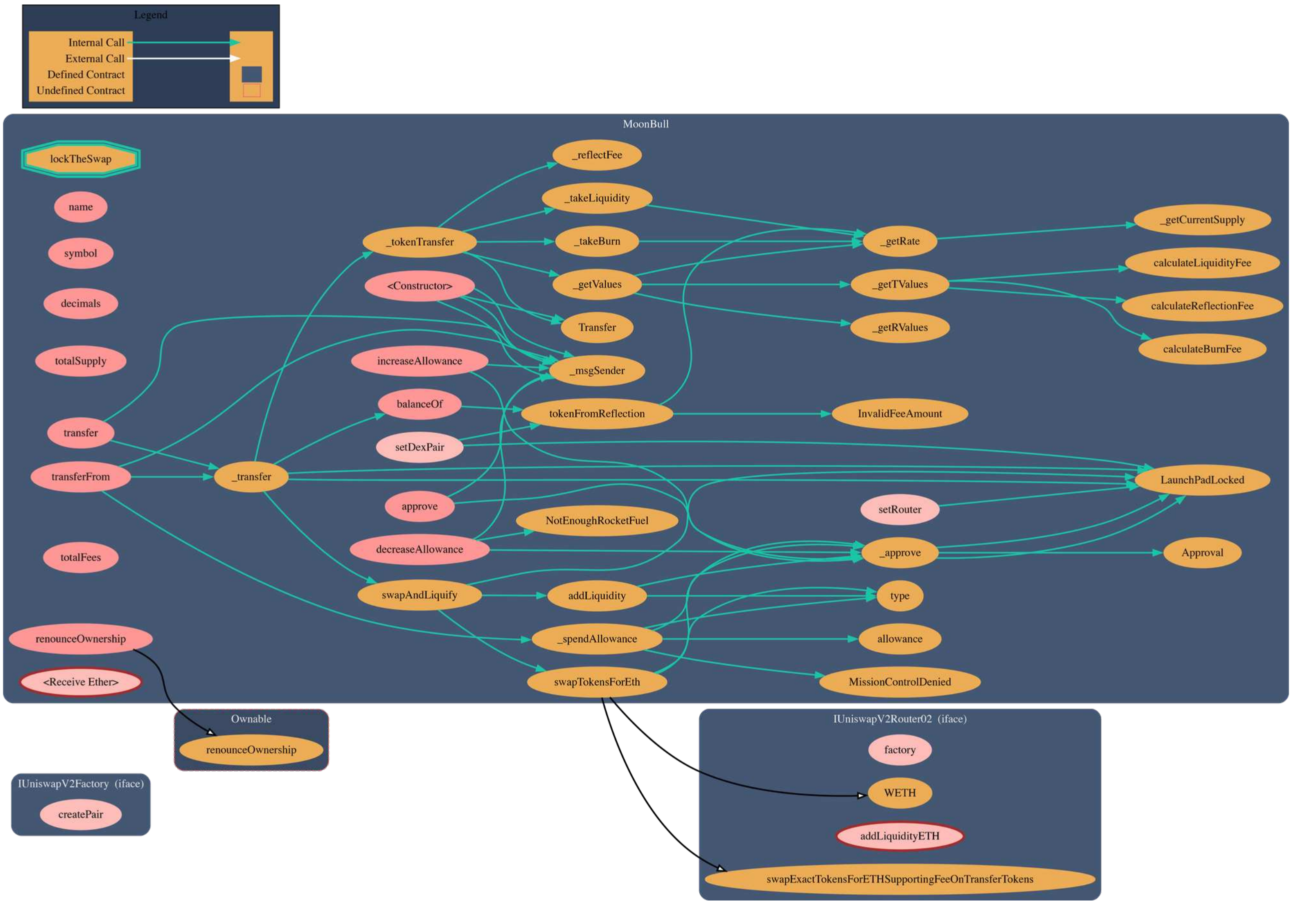
Contract	Type	Bases		
└				
Function Name	**Visibility**	**Mutability**	**Modifiers**	
IUniswapV2Factory	Interface			
└	createPair	External ❗️	🛑	NO❗️
IUniswapV2Router02	Interface			
└	factory	External ❗️		NO❗️
└	WETH	External ❗️		NO❗️
└	addLiquidityETH	External ❗️	💵	NO❗️
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗️	🛑	NO❗️
MoonBull	Implementation	Context, IERC20, IERC20Metadata, Ownable		
└		Public ❗️	🛑	Ownable
└	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❗️

Contract	Type	Bases		
└	totalFees	Public ❗		NO ❗
└	tokenFromReflection	Public ❗		NO ❗
└	setRouter	External ❗	🛑	onlyOwner
└	setDexPair	External ❗	🛑	onlyOwner
└	renounceOwnership	Public ❗	🛑	onlyOwner
└		External ❗	💵	NO ❗
└	_approve	Private 🔐	🛑	
└	_spendAllowance	Internal 🔒	🛑	
└	_transfer	Private 🔐	🛑	
└	swapAndLiquify	Private 🔐	🛑	lockTheSwap
└	swapTokensForEth	Private 🔐	🛑	
└	addLiquidity	Private 🔐	🛑	
└	_tokenTransfer	Private 🔐	🛑	
└	_reflectFee	Private 🔐	🛑	
└	_getValues	Private 🔐		
└	_getTValues	Private 🔐		
└	_getRValues	Private 🔐		
└	_getRate	Private 🔐		
└	_getCurrentSupply	Private 🔐		
└	_takeLiquidity	Private 🔐	🛑	
└	_takeBurn	Private 🔐	🛑	
└	calculateReflectionFee	Private 🔐		
└	calculateLiquidityFee	Private 🔐		
└	calculateBurnFee	Private 🔐		

Legend

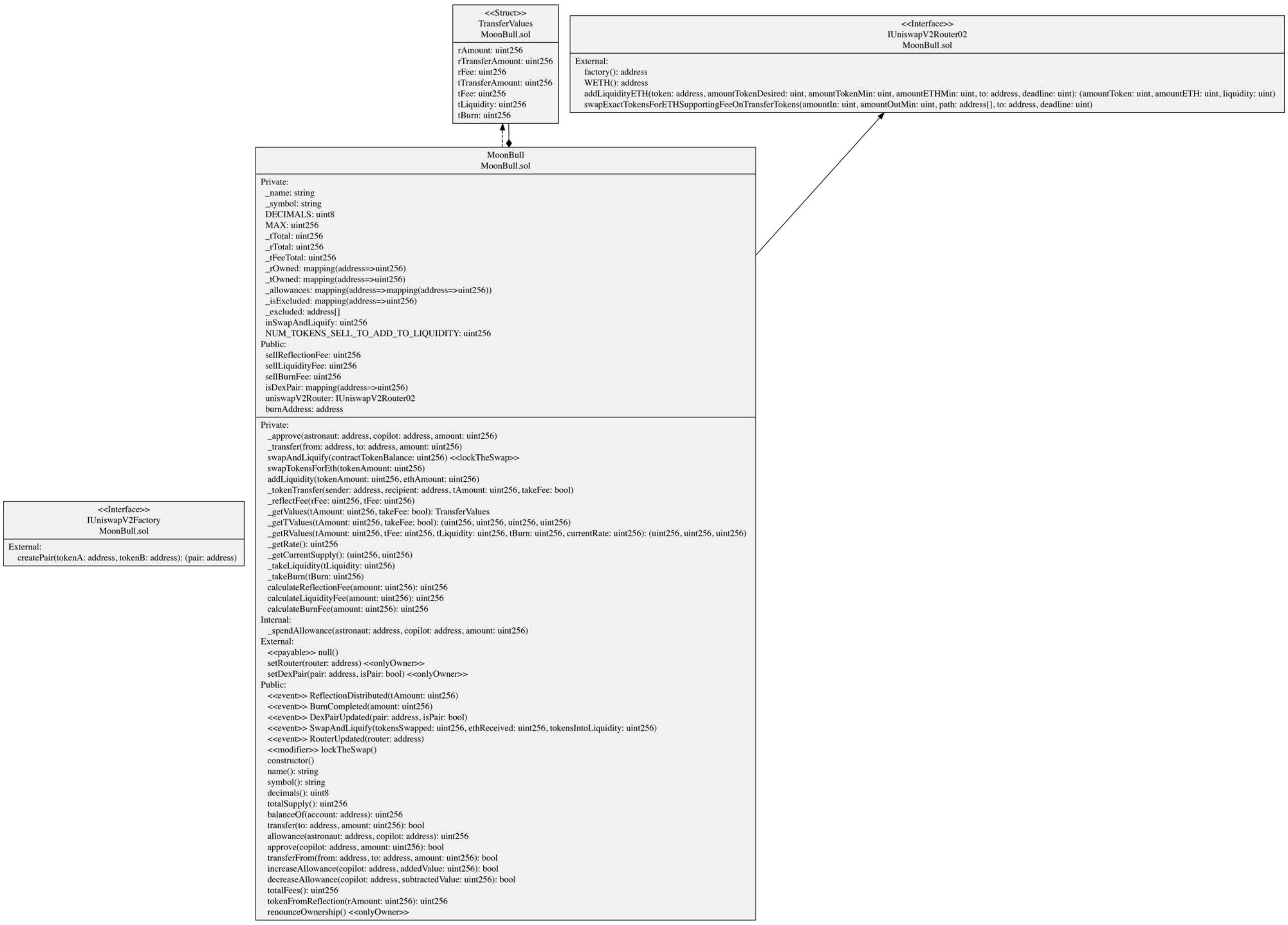
Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

A.4 Call Graph



Document ID: 19810CD20B84A5ABDACBEEA9B95E6D1D

A.5 UML Class Diagram



About SCRL

SCRL is a dynamic cybersecurity team that provides security assessments, penetration testing, and incident investigations. We utilize specialized tools, expertise, and frameworks to enhance security measures. Additionally, we develop robust internal tools tailored to our organization's needs.
SCRL is committed to driving security for the blockchain industry and businesses alike.

Follow Us On:

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