# Security Audit

## Full Audit Report For Apeing


**Client: Apeing**
https://www.apeing.com/

November 6, 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.

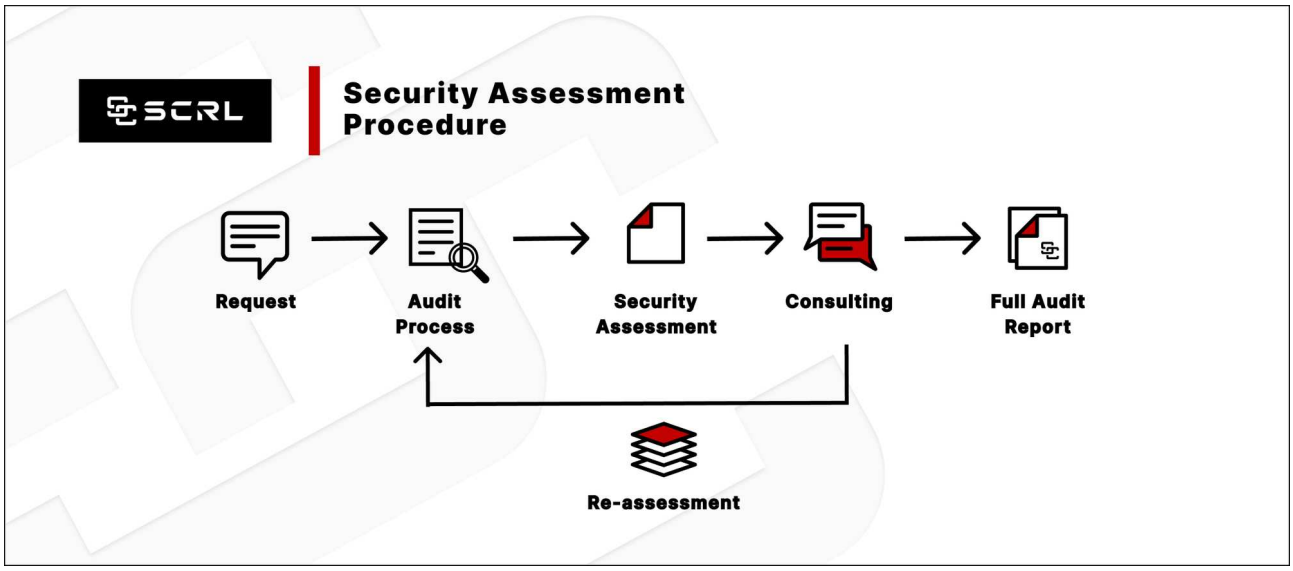
If the Service Provider identifies a vulnerability, we will notify the Service User via a Preliminary Report, which will be maintained in confidence for security purposes. The Service Provider disclaims responsibility in the event of any attacks occurring before or after conducting a security assessment. 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 **November 3, 2025**


| Audit Score | Project Website | X (Twitter) | Telegram |
|---|---|---|---|
| **9.4** | https://www.apeing.com/ | https://x.com/apeingcoin | https://t.me/apeingcoin |

| Client | Confidential | Audit Method | Language |
|---|---|---|---|
| Apeing | Private | Static Analysis + Dynamic Analysis + Manual Review + Formal Verification + Hyper Fuzzing + WAS (Custom Detector) | Solidity |

## Vulnerabilities Summary:

| 2 | 2 | 0 | 0 | 0 |
|---|---|---|---|---|
| **Total Findings** | **Resolved** | **Mitigated** | **Acknowledge** | **Declined** |

| | | | |
|---|---|---|---|
| 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. |
| 0 | Medium | | It essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract. |
| 0 | Low | | While low-severity issues can be less urgent, it still advisable to address them to improve the overall security posture of the smart contract. |
| 2 | Informational | Resolved: 2 | 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/Apeing.sol | d4557daa5a313cbbcf55beea22199407f3654705 |

**Onchain Scope:**

| Chain | Contract Address |
|---|---|
| Not Deployed | - |

**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:**

# 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 | 0 | 0 | 0 | 2 |
|---|---|---|---|---|
| Critical | High | Medium | Low | Informational |

This security assessment report for **Apeing** a total of **2 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 |
|---|---|---|---|---|
| **SNC-01** | Conformity to Solidity naming conventions | Informational | **Coding Style** | **Resolved** |
| **UIC-01** | Arithmetic Operations Not Wrapped in unchecked Block | Informational | **Gas Optimization** | **Resolved** |

# 8 Findings

## SNC-01: Conformity to Solidity naming conventions

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

### Finding Code

```
❌ Constant Apeing._decimals (src/Apeing.sol:33) is not in UPPER_CASE_WITH_UNDERSCORES
❌ Constant Apeing._initialSupply (src/Apeing.sol:30) is not in UPPER_CASE_WITH_UNDERSCORES
❌ Constant Apeing._name (src/Apeing.sol:31) is not in UPPER_CASE_WITH_UNDERSCORES
❌ Constant Apeing._symbol (src/Apeing.sol:32) is not in UPPER_CASE_WITH_UNDERSCORES
```

### 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

[2025-11-06] Apeing Team has already fixed this issue by refactoringthe the code

On line 30-33

```
    uint256 private constant _INITIAL_SUPPLY = 16_750_000_000 * 10**18;

    string private constant _NAME = "Apeing";

    string private constant _SYMBOL = "$APEING";

    uint8 private constant _DECIMALS = 18;
```

On line 41, 45, 49, 57

```
41:     _mint(_msgSender(), _INITIAL_SUPPLY);

45:     return _DECIMALS;

49:     return _SYMBOL;

57:     return _NAME;
```

### References

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

# UIC-01: Arithmetic Operations Not Wrapped in unchecked Block

| Vulnerabilities | Severity | Category | Status |
|---|---|---|---|
| **Arithmetic Operations Not Wrapped in unchecked Block** | Informational | Gas Optimization | Resolved |

## Finding Code

```
30: uint256 private constant _initialSupply = 16_750_000_000 * 10**18;

82: _approve(degenApe, spender, _allowances[degenApe][spender] + addedValue);

100: _approve(degenApe, spender, currentAllowance - subtractedValue);

132: _totalSupply += amount;

133: _balances[account] += amount;

148: _balances[from] = bananaBalance - amount;

149: _balances[to] += amount;

164: _approve(tokenOwner, spender, currentAllowance - amount);

174: _balances[account] = bananaBalance - amount;

175: _totalSupply -= amount;
```

## Description

The contract performs arithmetic operations (e.g., +, -, *) that are guaranteed not to overflow due to surrounding logic or bounds checking, but these operations are not enclosed within an unchecked block.

Since Solidity 0.8.0, all arithmetic operations include automatic overflow and underflow checks by default, which adds additional gas costs. When the operation is provably safe — such as incrementing a loop counter within a bounded range or subtracting a known smaller value — wrapping the expression in an unchecked block can reduce gas usage by avoiding unnecessary checks.

## Recommendation

Wrap arithmetic expressions in an unchecked block when:

- Overflow is provably impossible, and
- The operation is performed in gas-sensitive contexts, such as loops.

## Alleviation

[2025-11-06] Apeing Team has already fixed this issue by refactoringthe the code

On line 82-88

```
        uint256 currentAllowance = _allowances[degenApe][spender];

        uint256 newAllowance;

        unchecked {

            newAllowance = currentAllowance + addedValue;
```

```
        }
        if (newAllowance < currentAllowance) revert ApeSwingTooFar();
        _approve(degenApe, spender, newAllowance);
```

## References

**Solidity Docs – Unchecked Arithmetic:** *https://docs.soliditylang.org/en/latest/control-structures.html#checked-or-unchecked-arithmetic*

# A Appendix

## A.1 Hyper Fuzzing

| Function | Status |
|---|---|
| apeing_approve(address,uint256) | ✅ |
| apeing_burn(uint256) | ✅ |
| asset_approve(address,uint128) | ✅ |
| apeing_increaseAllowance(address,uint256) | ✅ |
| apeing_transfer(address,uint256) | ✅ |
| switch_asset(uint256) | ✅ |
| asset_mint(address,uint128) | ✅ |
| switchActor(uint256) | ✅ |
| add_new_asset(uint8) | ✅ |
| apeing_decreaseAllowance(address,uint256) | ✅ |
| apeing_burnFrom(address,uint256) | ✅ |
| apeing_transferFrom(address,address,uint256) | ✅ |
| AssertionFailed(..) | ✅ |

## A.2 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/Apeing.sol | 1 | *** | 205 | 180 | 124 | 28 | 95 | **Σ** |
| 📝 | **Totals** | **1** | *** | **205** | **180** | **124** | **28** | **95** | **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, ...)

# A.3 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 |
|---|---|
| 13 | 0 |

| External | Internal | Private | Pure | View |
|---|---|---|---|---|
| 13 | 18 | 0 | 3 | 3 |

### StateVariables

| Total | 🌐****Public |
|---|---|
| 7 | 0 |

### Capabilities

| Solidity Versions observed | 🧪** Experimental Features** | 💰** Can Receive Funds** | 📲** Uses Assembly** | 💣** Has Destroyable Contracts** |
|---|---|---|---|---|
| 0.8.20 | | *** | *** | *** |

| 📤** 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/token/ERC20/extensions/IERC20Metadata.sol | 1 |

| Dependency / Import Path | Count |
| --- | --- |
| @openzeppelin/contracts/utils/Context.sol | 1 |

## A.4 Contracts Description Table

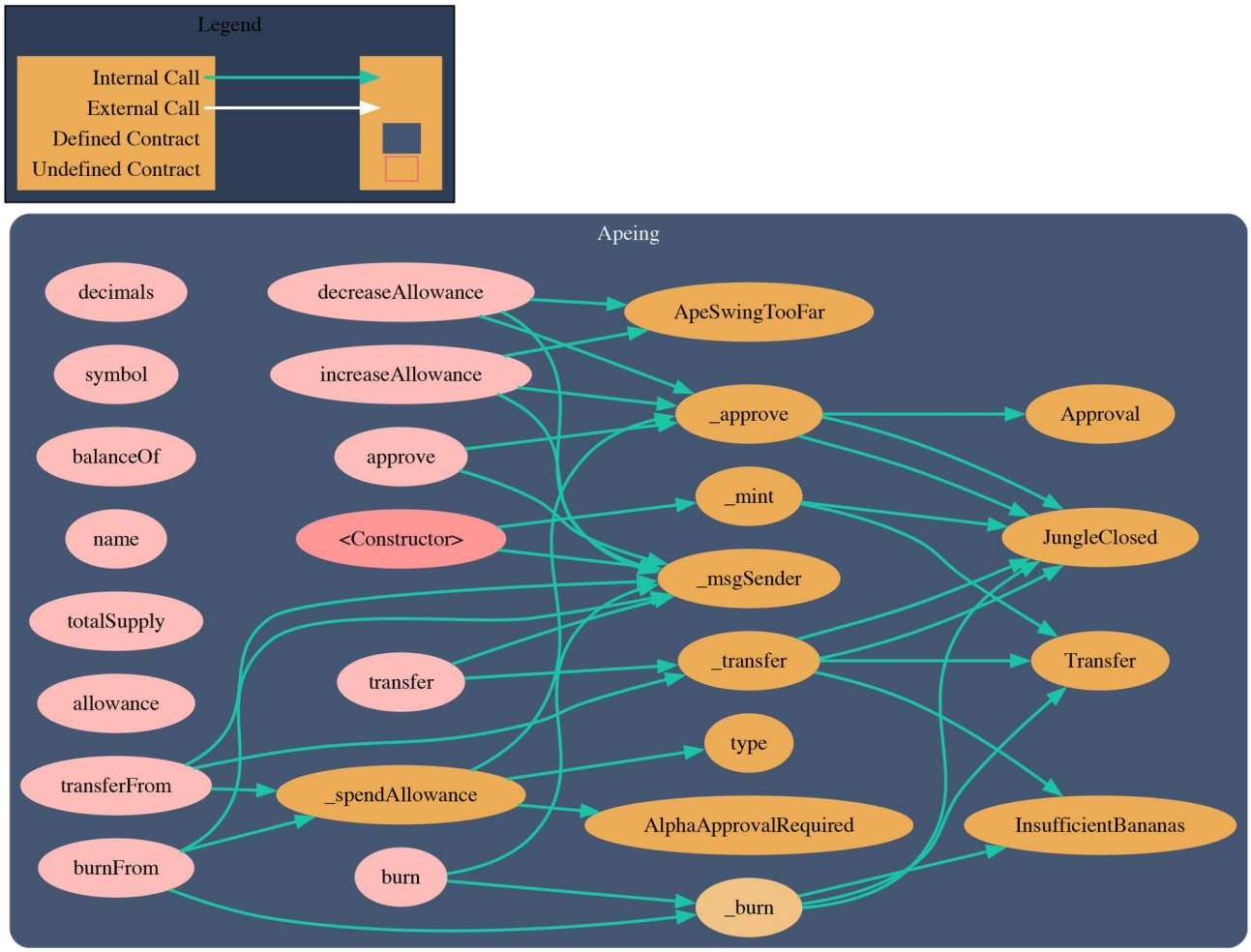
Contracts Description Table

| Contract | Type | Bases | | |
|---|---|---|---|---|
| └ | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **Apeing** | Implementation | Context, IERC20, IERC20Metadata | | |
| └ | | Public ❗ | 🛑 | NO ❗ |
| └ | decimals | External ❗ | | NO ❗ |
| └ | symbol | External ❗ | | NO ❗ |
| └ | balanceOf | External ❗ | | NO ❗ |
| └ | name | External ❗ | | NO ❗ |
| └ | approve | External ❗ | 🛑 | NO ❗ |
| └ | totalSupply | External ❗ | | NO ❗ |
| └ | allowance | External ❗ | | NO ❗ |
| └ | increaseAllowance | External ❗ | 🛑 | NO ❗ |
| └ | transfer | External ❗ | 🛑 | NO ❗ |
| └ | decreaseAllowance | External ❗ | 🛑 | NO ❗ |
| └ | transferFrom | External ❗ | 🛑 | NO ❗ |
| └ | _approve | Internal 🔒 | 🛑 | |
| └ | _mint | Internal 🔒 | 🛑 | |
| └ | _transfer | Internal 🔒 | 🛑 | |
| └ | _spendAllowance | Internal 🔒 | 🛑 | |
| └ | _burn | Internal 🔒 | 🛑 | |
| └ | burn | External ❗ | 🛑 | NO ❗ |
| └ | burnFrom | External ❗ | 🛑 | NO ❗ |

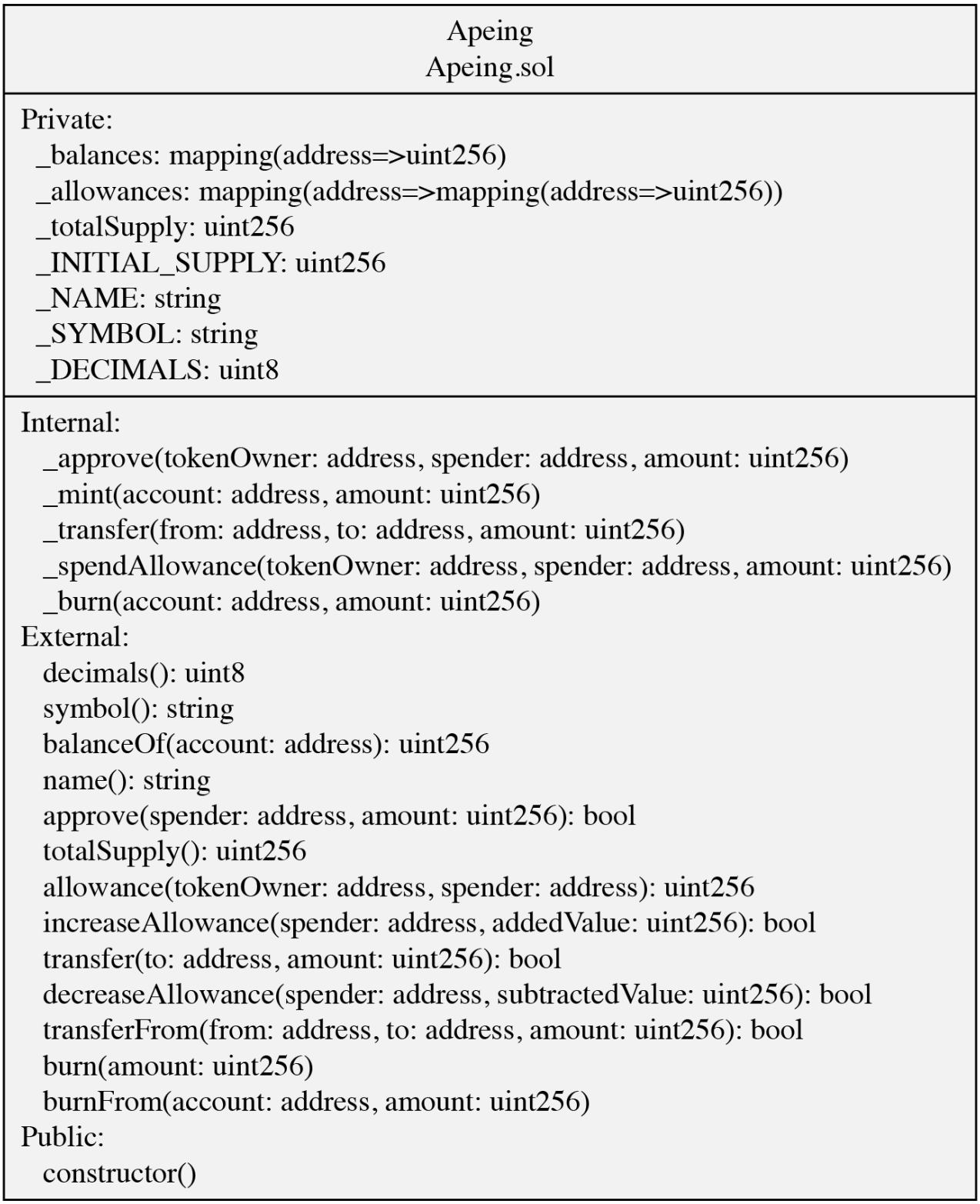
Legend

| Symbol | Meaning |
|---|---|
| 🛑 | Function can modify state |
| 💵 | Function is payable |

## A.5 Call Graph



## A.6 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/* |