



Full Audit Report

KEKW Security Assessment



KEKW Security Assessment
FULL AUDIT REPORT

Security Assessment by SCRL on **Tuesday, September 12, 2023**

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



Executive Summary

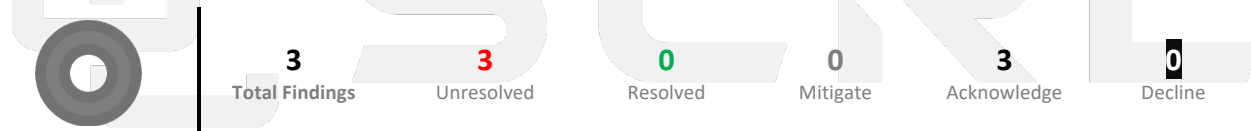
For this security assessment, SCRL received a request on Sunday, September 10, 2023

Client	Language	Audit Method	Confidential	Network Chain	Contract
KEKW	Solidity	Whitebox	Public	Ethereum	0x0DF596AD12F927e41EC317AF7DD666cA3574845f
Report Version	Twitter	Telegram	Website		
1.1	https://twitter.com/kek_w_gg	https://t.me/kek_w_gg	https://kek_w_gg/		

CVSS Scoring:



Vulnerability Summary



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's 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'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
contracts/KEKW.sol	3941cea41448ab76c297f125d92235896e96aa7f

Audit Version History:

Version	Date	Description
1.0	Sunday, September 10, 2023	Preliminary Report
1.1	Tuesday, September 12, 2023	Full Audit Report

Audit information:

Request Date	Audit Date	Re-assessment Date
Sunday, September 10, 2023	Sunday, September 10, 2023	-

Smart Contract Audit Summary

The graphic features the SCRL logo on the left. To its right, the text reads: 'SCRL has assessed the security of this smart contract. The results of the security assessment revealed No Critical Vulnerabilities.' Below this, it says 'Full Audit Report by SCRL on September 12, 2023'. On the right side of the graphic is a large red shield with a white checkmark inside a green circle, signifying a successful audit.

Security Assessment Author

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

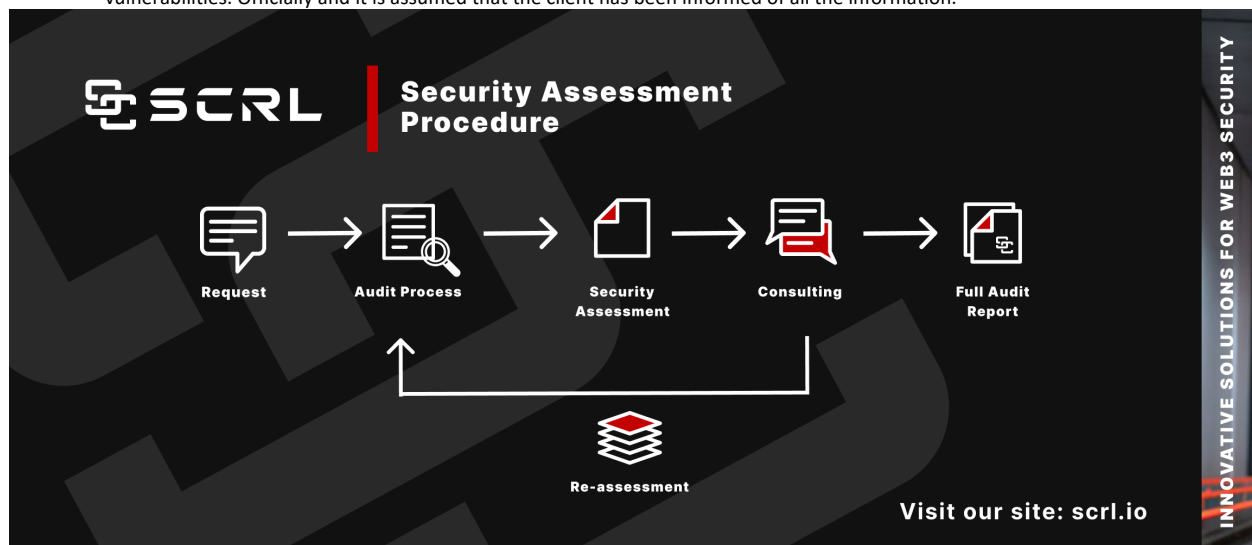
If the **service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security 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	Low	Medium	High
Impact [Likelihood]			
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



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 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

Summary

- Executive Summary
- CVSS Scoring
- Vulnerability Summary
- Audit Scope
- Audit Version History
- Audit Information
- Smart Contract Audit Summary
- Security Assessment Author
- Digital Sign
- Disclaimer
- Security Assessment Procedure
- Risk Rating
- Category

Source Code Detail

- Dependencies / External Imports
- Visibility, Mutability, Modifier function testing

Vulnerability Finding





- Vulnerability
- SWC Findings
- Contract Description
- Inheritance Relational Graph
- UML Diagram

About SCRL

Source Code Detail

Source Units Analyzed: 1

Source Units in Scope: 1 (100%)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/KEKW.sol	3	2	517	438	169	301	111	
	Totals	3	2	517	438	169	301	111	

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
2	0	2	1

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			
20	0			
External	Internal	Private	Pure	View
9	26	0	0	14

StateVariables

Total	 Public
5	0

Capabilities

<div>Solidity Versions observed</div>	<div> Experimental Features</div>	<div> Can Receive Funds</div>	<div> Uses Assembly</div>	<div> Has Destroyable Contracts</div>	
<div><div><div>^0.8.0</div><div>0.8.16</div></div></div>					
<div><div> Transfers ETH</div></div>	<div><div> Low-Level Calls</div></div>	<div><div> DelegateC all</div></div>	<div><div> Uses Hash Functions</div></div>	<div><div> ECRcov er</div></div>	<div><div> New/Create/C reate2</div></div>

 TryCatch	Σ Unchecked
<input type="text"/>	<input type="text" value="yes"/>

 SCRL

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
SEC-01	Conformance to numeric notation best practices (too-many-digits)	Informational	Best Practices	Acknowledge
SEC-02	Empty Function Body - Consider commenting why	Informational	Best Practices	Acknowledge
GAS-01	Use Custom Errors	Gas-optimization	Gas Optimization	Acknowledge

SEC-01: Conformance to numeric notation best practices (too-many-digits)

Vulnerability Detail	Severity	Location	Category	Status
Conformance to numeric notation best practices (too-many-digits)	Informational	Check on finding	Best Practices	Acknowledge

Finding:

```
✗ KEKW.constructor() (KEKW.sol:514-516) uses literals with too many digits:  
  • _mint(msg.sender,4206900000000000 * 1e18) (KEKW.sol#515)
```

Recommendation:

Use:

- [Ether suffix](<https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#ether-units>),
- [Time suffix](<https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#time-units>), or
- [The scientific notation](<https://solidity.readthedocs.io/en/latest/types.html#rational-and-integer-literals>)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits>

Alleviation:

KEKW has Acknowledge this issue.

SEC-02: Empty Function Body - Consider commenting why

Vulnerability Detail	Severity	Location	Category	Status
Empty Function Body - Consider commenting why	Informational	Check on finding	Best Practices	Acknowledge

Finding:

```
488:     function _beforeTokenTransfer(address from, address to, uint256 amount)
internal virtual {}

504:     function _afterTokenTransfer(address from, address to, uint256 amount)
internal virtual {}

...
```

Recommendation:

Commenting empty function bodies with an explanation of why they are empty is a good practice for maintaining clean and understandable Solidity code.

Alleviation:

KEKW has Acknowledge this issue.

GAS-01: Use Custom Errors

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

Finding:

```
340:         require(currentAllowance >= subtractedValue, "ERC20: decreased allowance
below zero");

363:         require(from != address(0), "ERC20: transfer from the zero address");

364:         require(to != address(0), "ERC20: transfer to the zero address");

369:         require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");

392:         require(account != address(0), "ERC20: mint to the zero address");

418:         require(account != address(0), "ERC20: burn from the zero address");

423:         require(accountBalance >= amount, "ERC20: burn amount exceeds balance");

449:         require(owner != address(0), "ERC20: approve from the zero address");

450:         require(spender != address(0), "ERC20: approve to the zero address");

467:         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.

[Source](<https://blog.soliditylang.org/2021/04/21/custom-errors/>)

Alleviation:

KEKW has Acknowledge this issue.







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

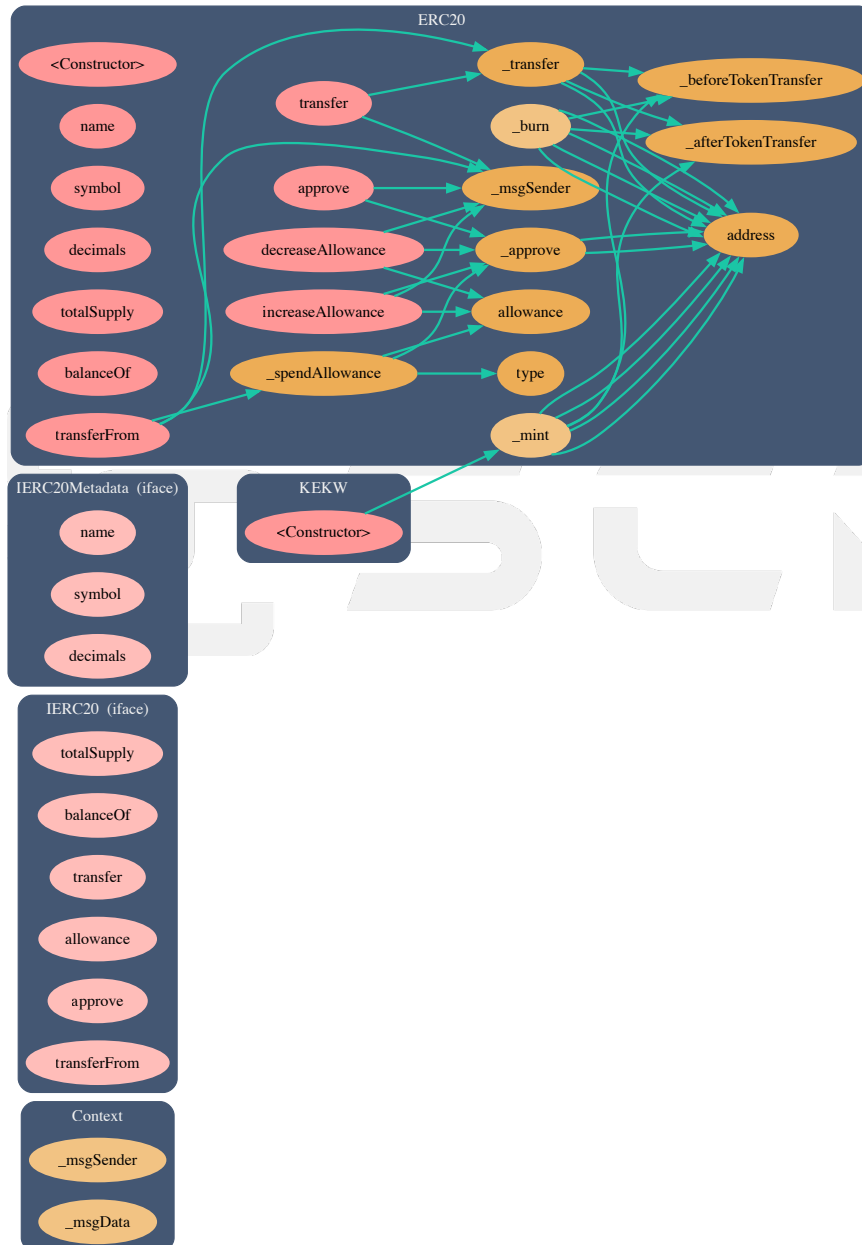
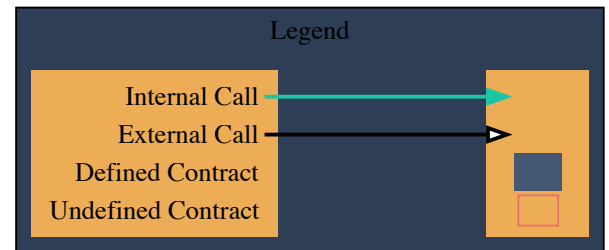
Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
L	_msgSender	Internal 		
L	_msgData	Internal 		
IERC20	Interface			
L	totalSupply	External !		NO !
L	balanceOf	External !		NO !
L	transfer	External !		NO !
L	allowance	External !		NO !
L	approve	External !		NO !
L	transferFrom	External !		NO !
IERC20Metadata	Interface	IERC20		
L	name	External !		NO !
L	symbol	External !		NO !
L	decimals	External !		NO !
ERC20	Implementation	Context, IERC20, IERC20Metadata		
L		Public !		NO !
L	name	Public !		NO !

Contract	Type	Bases		
L	symbol	Public !		NO !
L	decimals	Public !		NO !
L	totalSupply	Public !		NO !
L	balanceOf	Public !		NO !
L	transfer	Public !		NO !
L	allowance	Public !		NO !
L	approve	Public !		NO !
L	transferFrom	Public !		NO !
L	increaseAllowance	Public !		NO !
L	decreaseAllowance	Public !		NO !
L	_transfer	Internal 		
L	_mint	Internal 		
L	_burn	Internal 		
L	_approve	Internal 		
L	_spendAllowance	Internal 		
L	_beforeTokenTransfer	Internal 		
L	_afterTokenTransfer	Internal 		
KEKW	Implementation	ERC20		
L		Public !		ERC20

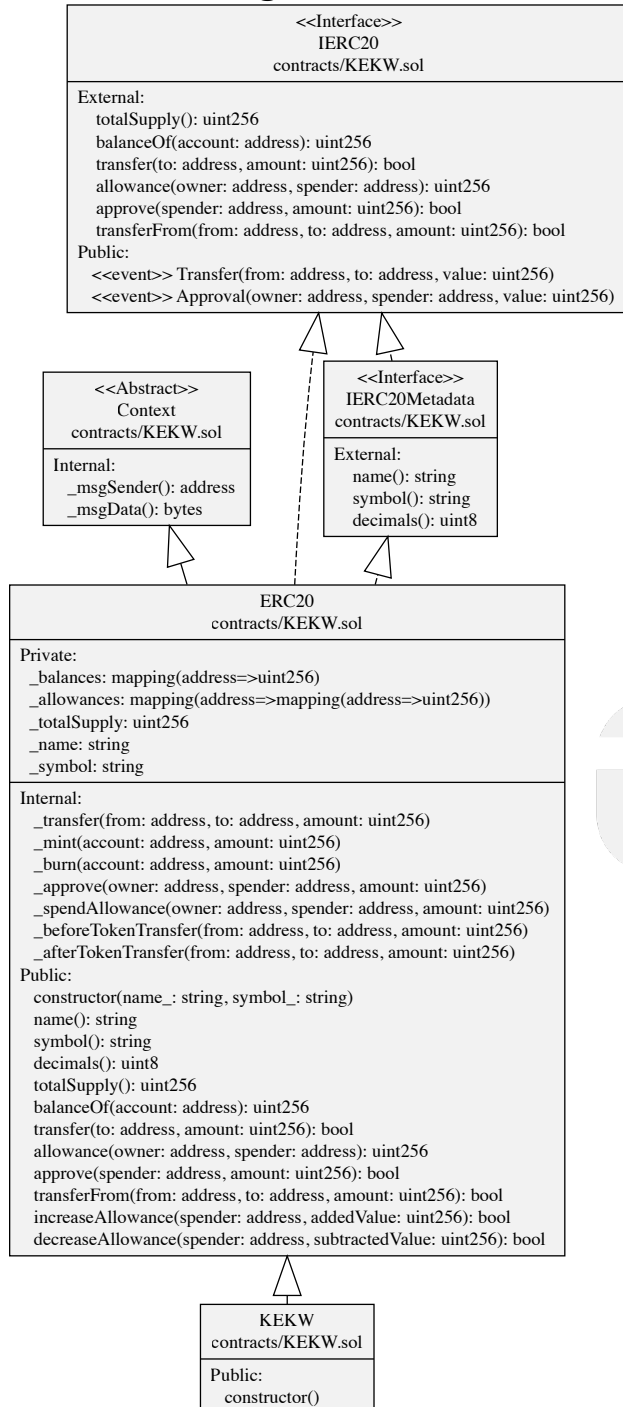
Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Inheritate Function Relation Graph

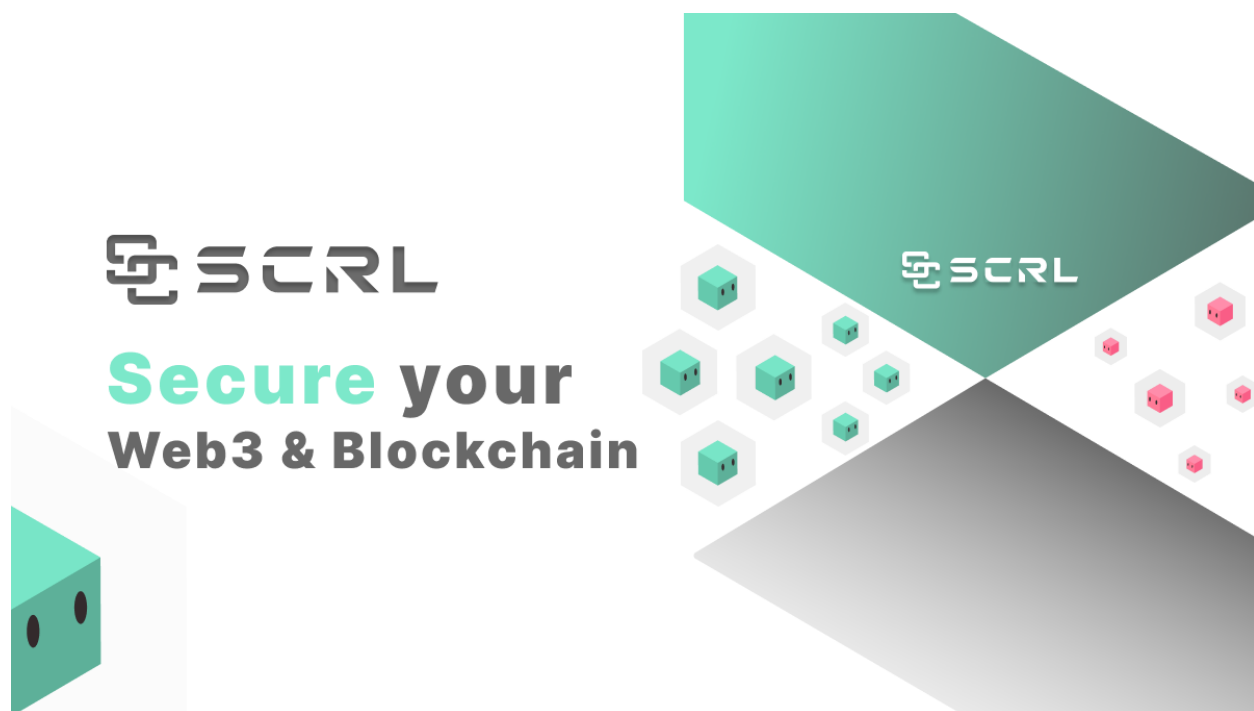


UML Class Diagram



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/