SCRL

Full Audit Report With re-assessment

DoubleUP Token Security Assessment

DoubleUP Token Security Assessment

FULL AUDIT REPORT WITH RE-ASSESSMENT

Security Assessment by SCRL on **Wednesday, May 15, 2024**

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

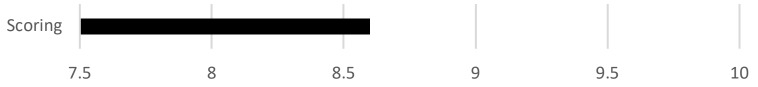
Executive Summary

For this security assessment, SCRL received a request on Sunday, May 5, 2024

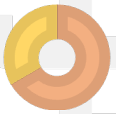
Client	Language	Audit Method	Confidential	Network Chain	Contract
DoubleUp Token	**Solidity**	**Whitebox**	**Public**	**Polygon Mainnet**	0x3667125d0D3f23EDf49DCF506a83A147C80ae1BA

Report Version	Twitter	Telegram	Website
1.1	https://twitter.com/doubleup_org	https://t.me/doubleup_org	https://doubleup.org/

Scoring:



Vulnerability Summary



2	0	2	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 Resolved**	It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
1	Low	**1 Resolved**	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.
0	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
0	Gas-optimization		Suggestions for more efficient algorithms or improvements in gas usage, even if the current code is already secure.

Audit Scope:

File	SHA-1 Hash
DBLU.sol	ed3a68e6ababbe223fb4fe632569fce275da124c

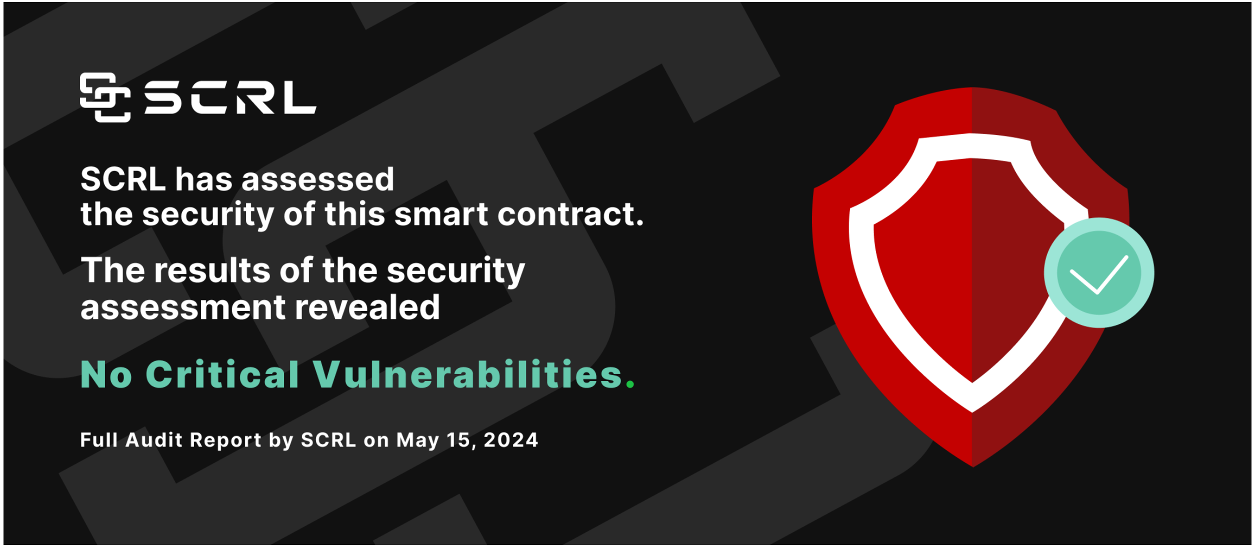
Audit Version History:

Version	Date	Description
1.0	Tuesday, 7 May, 2024	Preliminary Report
1.1	Wednesday, 15 May R 2024	Full Report With Re-assessment after deployed on mainnet

Audit information:

Request Date	Audit Date	Re-assessment Date
Sunday, May 5, 2024	Tuesday, May 7 2024	Wednesday, May 15, 2024

Smart Contract Audit Summary



SCRL has assessed
the security of this smart contract.

The results of the security
assessment revealed

No Critical Vulnerabilities.

Full Audit Report by SCRL on May 15, 2024

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

Digitally signed by Chinnakit J.
DN: cn=Chinnakit J., o=SCRL,
ou=CEO & Founder,
email=contact@scrl.io, c=TH
Date: 2024.05.15 19:04:42
+07'00'

Disclaimer

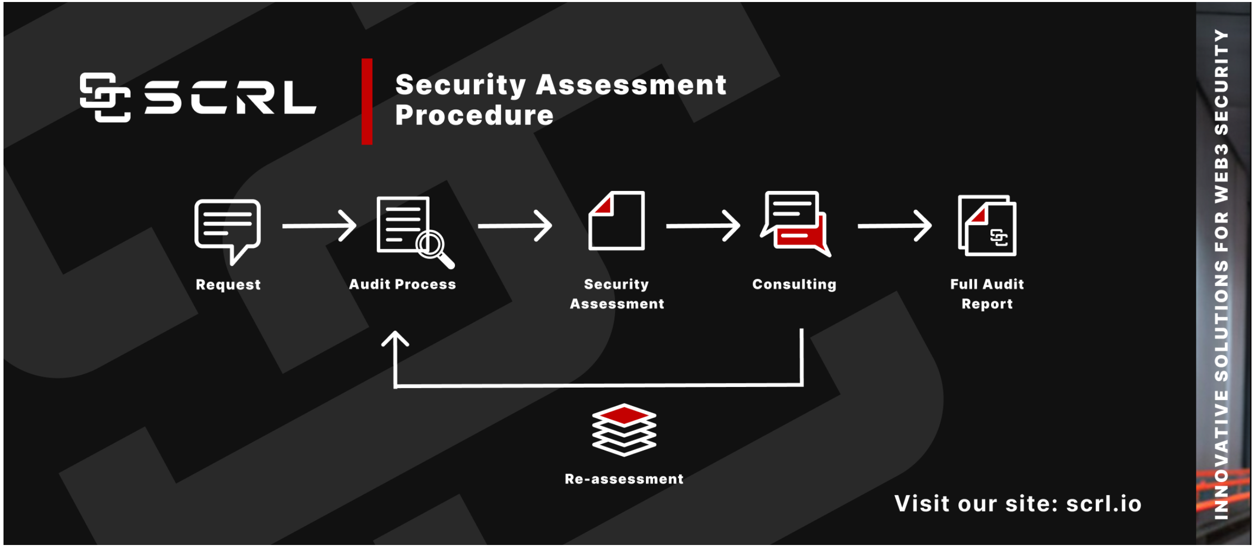
Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as "**Source code**".
And **SCRL** hereinafter referred to as "**Service Provider**", the **Service Provider** will not be held liable for any 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

Summary

Source Code Detail

Vulnerability Finding

About SCRL

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
📝📚 🔍🎨	DBLU.sol	6	1	787	736	240	457	174	🔆
📝📚 🔍🎨	**Totals**	**6**	**1**	**787**	**736**	**240**	**457**	**174**	🔆

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	1	1	3

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
23	0

External	Internal	Private	Pure	View
6	43	0	13	12

StateVariables

Total	🌐Public
15	1

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
>=0.6.0 <0.8.0 0.7.6				

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2

♻️ TryCatch	Σ Unchecked

Dependencies / External Imports

Dependency / Import Path	Count

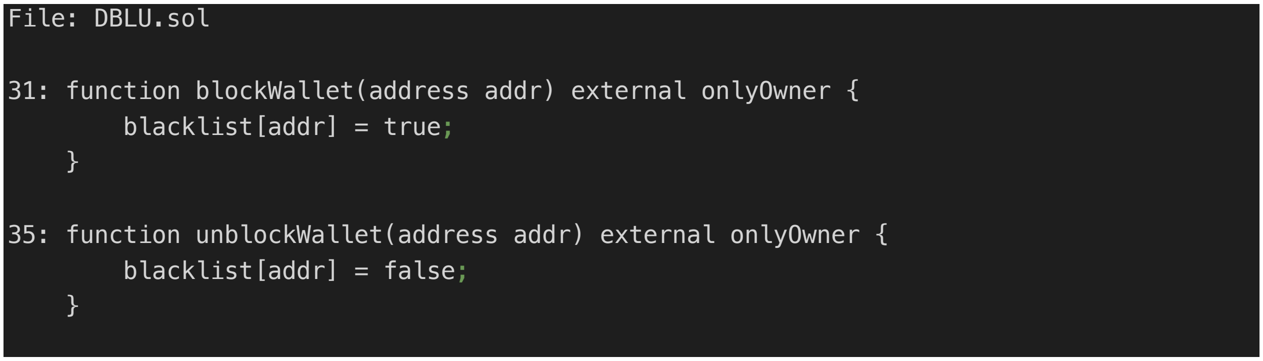
Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
CEN-01	Centralization Risk	Medium	Centralization	**Resolved**
SEC-01	beforeTokenTransfer function does not follow OZ documentation	Low	Security Risk	**Resolved**

CEN-01: Centralization Risk

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

Finding:



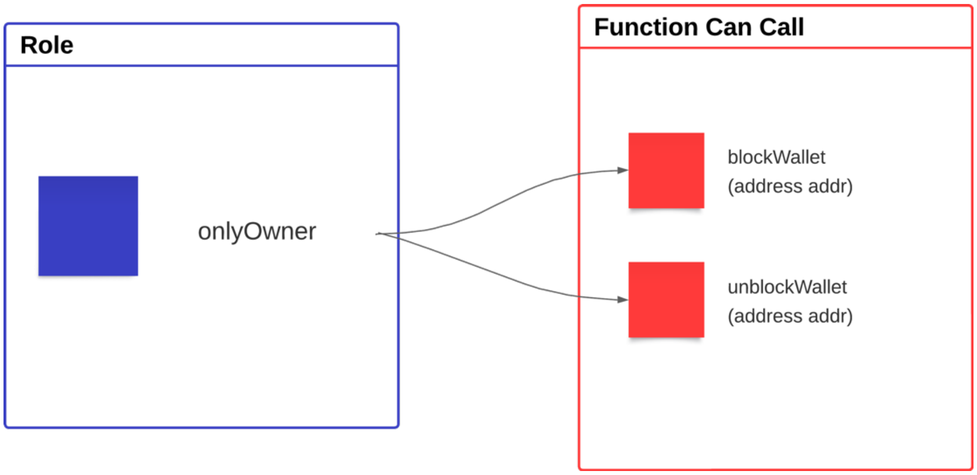
```
File: DBLU.sol

31: function blockWallet(address addr) external onlyOwner {
        blacklist[addr] = true;
    }

35: function unblockWallet(address addr) external onlyOwner {
        blacklist[addr] = false;
    }
```

Explain Function Capability:

The contract provides several functions:

1. **blockWallet(address addr)**
   - This function allows the contract owner to block a specific wallet address (**addr**).
   - When invoked, it sets the corresponding entry in the **blacklist** mapping to **true**, indicating that the specified wallet is blocked from certain actions or functionalities within the contract.
   - This capability is useful for implementing restrictions or sanctions on certain addresses, perhaps due to suspicious activity or violations of contract rules.
2. **unblockWallet(address addr)**
   - This function allows the contract owner to unblock a previously blocked wallet address (**addr**).
   - When invoked, it sets the corresponding entry in the **blacklist** mapping to **false**, effectively removing the block on that wallet.
   - This capability provides a way to reverse the actions taken by the **blockWallet** function, restoring the ability of the previously blocked wallet to participate in contract interactions.

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.

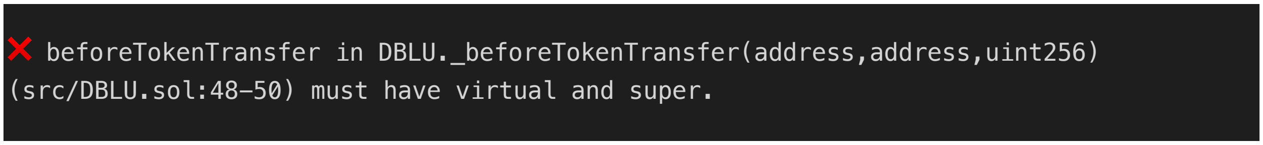
Alleviation:

The DoubleUp Team has fixed this issue by removing those function.

SEC-01: beforeTokenTransfer function does not follow OZ documentation

Vulnerability Detail	Severity	Location	Category	Status
beforeTokenTransfer function does not follow OZ documentation	Low	Check on finding	Naming Conventions	**Resolved**

Finding:



```
✗ beforeTokenTransfer in DBLU._beforeTokenTransfer(address,address,uint256)
(src/DBLU.sol:48-50) must have virtual and super.
```

Recommendation:

Make sure that beforeTokenTransfer function is used in the correct way.

Reference: https://docs.openzeppelin.com/contracts/4.x/extending-contracts#rules_of_hooks

Alleviation:

The DoubleUp Team has fixed 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**		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└		Internal 🔒	🛑	
└	owner	Public ❗		NO ❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
SafeMath	Library			
└	tryAdd	Internal 🔒		
└	trySub	Internal 🔒		
└	tryMul	Internal 🔒		
└	tryDiv	Internal 🔒		
└	tryMod	Internal 🔒		
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		

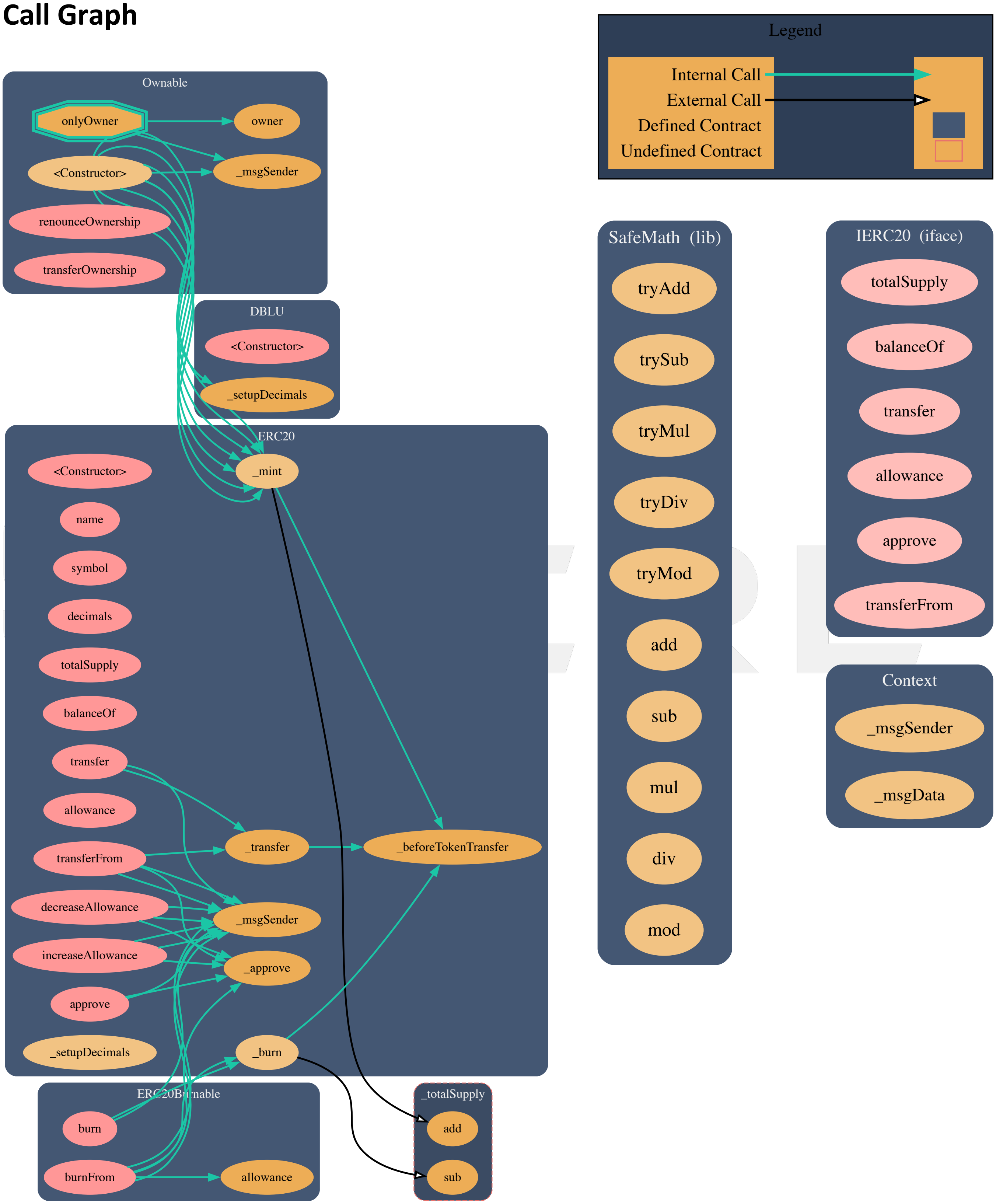
Contract	Type	Bases		
└	mod	Internal 🔒		
└	sub	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
ERC20	Implementation	Context, IERC20		
└		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❗

Contract	Type	Bases		
└	transferFrom	Public ❗	🛑	NO❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_setupDecimals	Internal 🔒	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
ERC20Burnable	Implementation	Context, ERC20		
└	burn	Public ❗	🛑	NO❗
└	burnFrom	Public ❗	🛑	NO❗
DBLU	Implementation	ERC20Burnable, Ownable		
└		Public ❗	🛑	ERC20

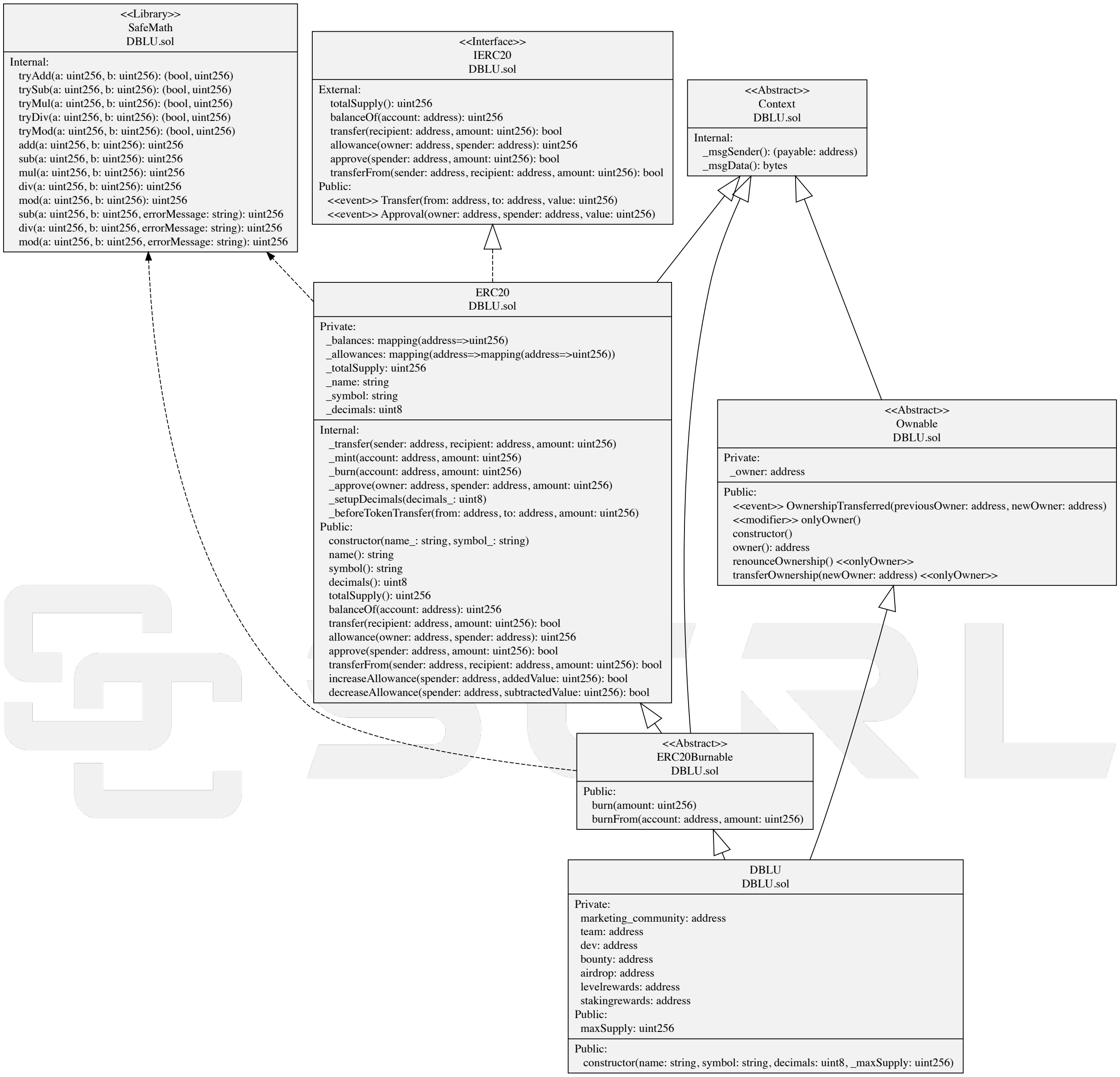
Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Call Graph

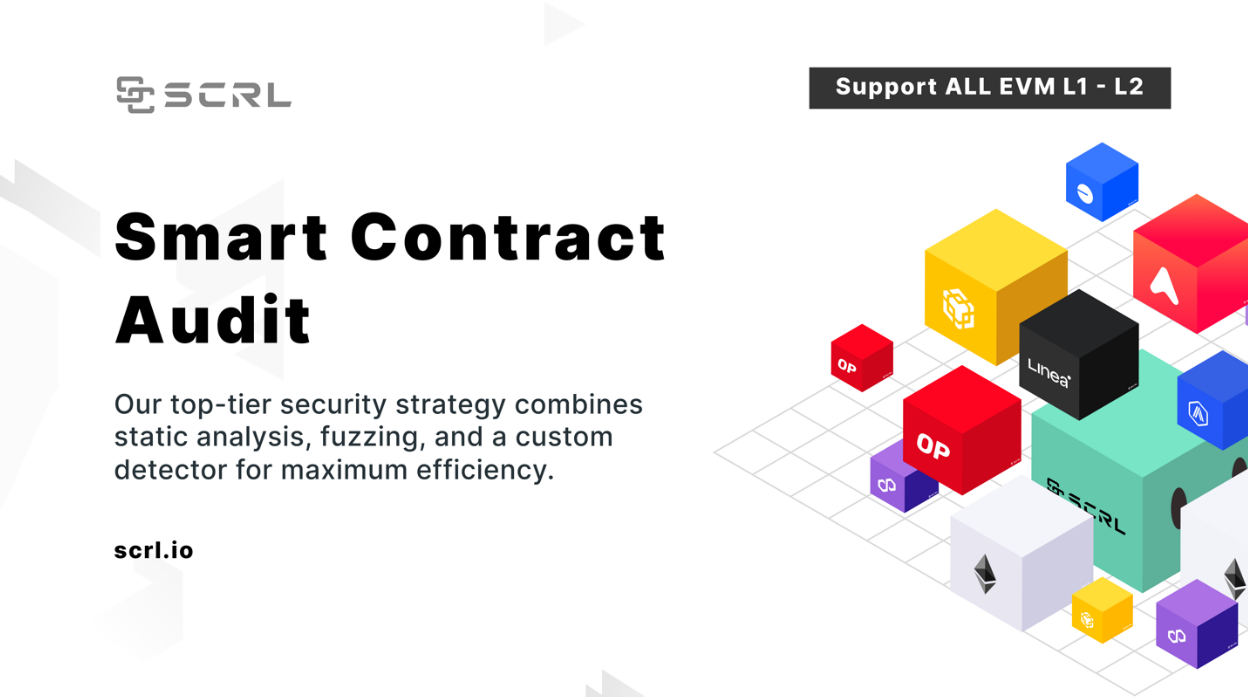

Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
transferOwnership
DBLU
<Constructor>
_setupDecimals
ERC20
<Constructor>
_mint
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
_transfer
_beforeTokenTransfer
transferFrom
decreaseAllowance
_msgSender
increaseAllowance
_approve
approve
_setupDecimals
_burn
ERC20Burnable
burn
burnFrom
allowance
_totalSupply
add
sub
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

UML Class Diagram


<<Library>>
SafeMath
DBLU.sol
Internal:
tryAdd(a: uint256, b: uint256): (bool, uint256)
trySub(a: uint256, b: uint256): (bool, uint256)
tryMul(a: uint256, b: uint256): (bool, uint256)
tryDiv(a: uint256, b: uint256): (bool, uint256)
tryMod(a: uint256, b: uint256): (bool, uint256)
add(a: uint256, b: uint256): uint256
sub(a: uint256, b: uint256): uint256
mul(a: uint256, b: uint256): uint256
div(a: uint256, b: uint256): uint256
mod(a: uint256, b: uint256): uint256
sub(a: uint256, b: uint256, errorMessage: string): uint256
div(a: uint256, b: uint256, errorMessage: string): uint256
mod(a: uint256, b: uint256, errorMessage: string): uint256
<<Interface>>
IERC20
DBLU.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Abstract>>
Context
DBLU.sol
Internal:
_msgSender(): (payable: address)
_msgData(): bytes
ERC20
DBLU.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
_decimals: uint8
Internal:
_transfer(sender: address, recipient: address, amount: uint256)
_mint(account: address, amount: uint256)
_burn(account: address, amount: uint256)
_approve(owner: address, spender: address, amount: uint256)
setupDecimals(decimals: uint8)
_beforeTokenTransfer(from: address, to: address, amount: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
increaseAllowance(spender: address, addedValue: uint256): bool
decreaseAllowance(spender: address, subtractedValue: uint256): bool
<<Abstract>>
Ownable
DBLU.sol
Private:
_owner: address
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Abstract>>
ERC20Burnable
DBLU.sol
Public:
burn(amount: uint256)
burnFrom(account: address, amount: uint256)
DBLU
DBLU.sol
Private:
marketing_community: address
team: address
dev: address
bounty: address
airdrop: address
levelrewards: address
stakingrewards: address
Public:
maxSupply: uint256
Public:
constructor(name: string, symbol: string, decimals: uint8, _maxSupply: uint256)

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.



Support ALL EVM L1 - L2

Smart Contract Audit

Our top-tier security strategy combines static analysis, fuzzing, and a custom detector for maximum efficiency.

scrl.io

Follow Us On:

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