SCRL

Full Audit Report

X-light Security Assessment

X-light Security Assessment

FULL AUDIT REPORT

Security Assessment by SCRL on **Friday, August 4, 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 Friday, August 4, 2023

Client	Language	Audit Method	Confidential	Network Chain	Contract
X-LIGHT	**Solidity**	**Whitebox**	**Public**	**Base chain**	**0x33E646c84e289e5DFaFD6D61B9875af68089F40f**

Report Version	Twitter	Telegram	Website
1.1	**https://twitter.com/xlight_base**	**https://t.me/xlightbase**	**https://xlight.world/**

CVSS Scoring:

Score is 7.6

Scoring 0 1 2 3 4 5 6 7 8 9 10

Vulnerability Summary

7	7	0	0	7	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.
1	High	1 Unresolved	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.
4	Low	4 Unresolved	While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.
0	Very Low		Very Low severity is used for minor security concerns that have minimal impact and are generally of low risk.
1	Informational	1 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:

Cotract	Link
0x33E646c84e289e5DFaFD6D61B9875af68089F40f	https://basescan.org/token/0x33E646c84e289e5DFaFD6D61B9875af68089F40f#code

Audit Version History:

Version	Date	Description
1.0	Friday, August 4, 2023	Preliminary Report
1.1	Friday, August 4, 2023	Full Audit Report

Audit information:

Request Date	Audit Date	Re-assessment Date
Friday, August 4, 2023	Friday, August 4, 2023	-

Smart Contract Audit Summary

Security Assessment Author

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

Digital Sign

Digitally signed by CHIN J
DN: cn=CHIN J, o=SECURI LAB - Cybersecurity | Audit | KYC, ou=CEO, email=contact@securi-lab.com, c=TH
Date: 2023.08.04 22:07:20 +07'00'

Disclaimer

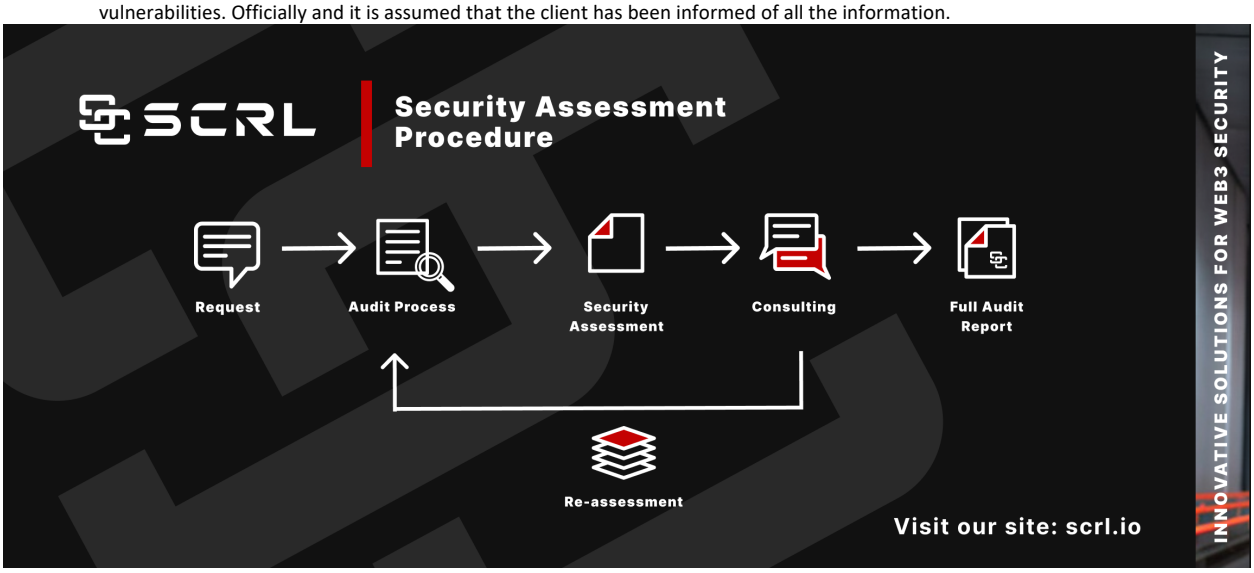
Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as "**Source code**".
And **SCRL** hereinafter referred to as "**Service Provider**", the **Service Provider** will not be held liable for any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

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/xlight.sol	6	4	1710	1203	742	402	492	🖥️💰📩👥🔆Σ
📝📚🔍🎨	**Totals**	**6**	**4**	**1710**	**1203**	**742**	**402**	**492**	🖥️💰📩👥🔆Σ

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Visibility, Mutability, Modifier function testing

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
1	2	4	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
66	6

External	Internal	Private	Pure	View
42	85	23	20	34

StateVariables

Total	🌐Public
27	8

Capabilities

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

`^0.8.0`
`^0.8.1`
`>=0.6.2`
`>=0.5.0`
`=0.8.4`
		`yes`	`yes`
			(1 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 Delegate Call	🧮 Uses Hash Functions	🧹 ECRecover	🌀 New/Create/Create2
yes		yes			

♻️ TryCatch	Σ Unchecked
	yes

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
SEC-01	Centralization Risk	High	Centralization	**Acknowledge**
SEC-02	Missing Zero Address Validation (missing-zero-check)	Low	Best Practices	**Acknowledge**
SEC-03	Missing Events Arithmetic (events-maths)	Low	Best Practices	**Acknowledge**
SEC-04	Reentrancy vulnerabilities leading to out-of-order Events (reentrancy-events)	Low	Best Practices	**Acknowledge**
SEC-05	Benign reentrancy vulnerabilities (reentrancy-benign)	Low	Best Practices	**Acknowledge**
SEC-06	Avoid using block timestamp	Informational	Best Practices	**Acknowledge**
GAS-01	Use Custom Errors	Gas-optimization	Gas Optimization	**Acknowledge**

SEC-01: Centralization Risk

Vulnerability Detail	Severity	Location	Category	Status
Centralization Risk	High	Check on finding	Centralization	Acknowledge

Finding:

```
989: contract LiquidityGeneratorToken is IERC20, Ownable, BaseToken {

1259:     function excludeFromReward(address account) public onlyOwner {

1269:     function includeInReward(address account) external onlyOwner {

1306:     function excludeFromFee(address account) public onlyOwner {

1310:     function includeInFee(address account) public onlyOwner {

1314:     function setTaxFeePercent(uint256 taxFeeBps) external onlyOwner {

1333:     function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
```

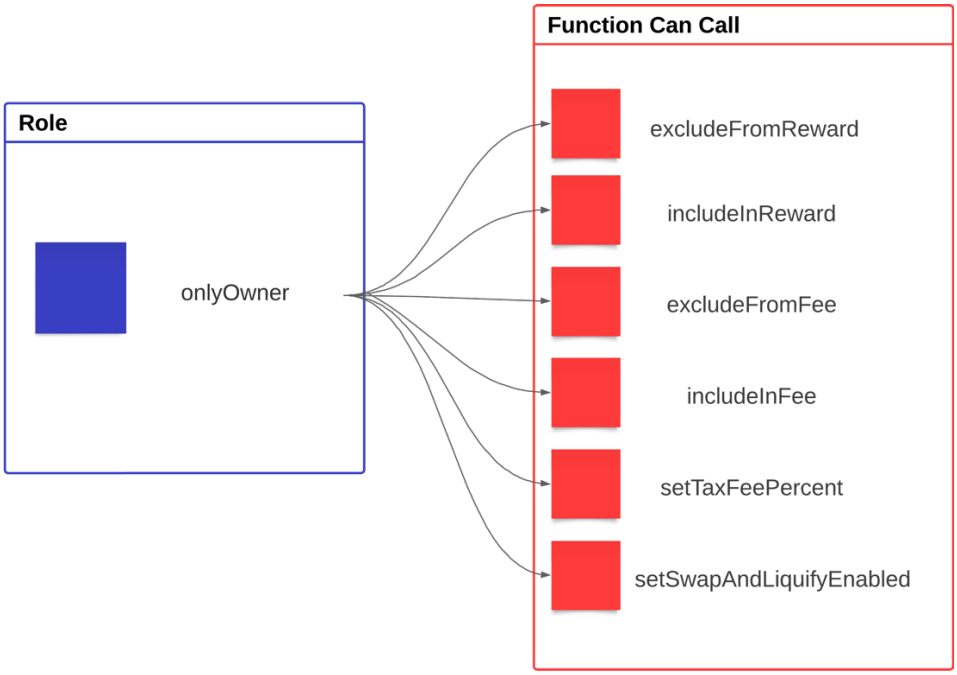
Explain Function Capability:

The contract provides several functions:

1. **excludeFromReward:** This function allows the contract owner to exclude an address from receiving rewards (reflections). If the contract owner has the sole authority to exclude addresses, it can lead to centralization risk as they could favor certain addresses or manipulate the reward distribution.
2. **includeInReward:** This function allows the contract owner to include an address in receiving rewards. Similar to excludeFromReward, the ability to control reward distribution can be a centralization risk if it's not done transparently and democratically.
3. **excludeFromFee:** This function allows the contract owner to exclude an address from paying fees on transactions. Again, if the contract owner has exclusive control over this, it can lead to centralization risk, favoring certain addresses or groups over others.
4. **includeInFee:** This function allows the contract owner to include an address in paying fees on transactions. Similar to excludeFromFee, the centralization risk arises if the owner can arbitrarily decide which addresses pay fees.
5. **setTaxFeePercent:** This function allows the contract owner to set the tax fee percentage. The tax fee is deducted from each transaction and often contributes to liquidity and rewards. Centralization risk emerges if the owner can unilaterally set and change the tax fee without community consensus.
6. **setSwapAndLiquifyEnabled:** This function allows the contract owner to enable or disable the swap and liquify functionality. This functionality is used to generate liquidity by swapping tokens for ETH and adding liquidity to a pool. Centralization risk arises if the owner can **control this feature without community input, as it could impact token liquidity and price stability.**

Please check to Contracts Description Table section to see full table of contract

Centralization Risk Contract **LiquidityGeneratorToken**



In the **LiquidityGeneratorToken** contract, Owner can call functions **excludeFromReward, includeInReward, excludeFromFee, includeInFee, setTaxFeePercent, setSwapAndLiquifyEnabled**

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:
X-light Team has Acknowledge this issue.

SEC-02: Missing Zero Address Validation (missing-zero-check)

Vulnerability Detail	Severity	Location	Category	Status
Missing Zero Address Validation (missing-zero-check)	Low	Check on finding	Best Practices	**Acknowledge**

Finding:

❌
LiquidityGeneratorToken.constructor(string,string,uint256,address,address,uint16,uint16,uint16,address,uint256).serviceFeeReceiver_ (xlight.sol:1053) lacks a zero-check on :

- address(serviceFeeReceiver_).transfer(serviceFee_) (xlight.sol#1114)

Recommendation:

Check that the address is not zero.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Alleviation:

X-light Team has Acknowledge this issue.

SEC-03: Missing Events Arithmetic (events-maths)

Vulnerability Detail	Severity	Location	Category	Status
Missing Events Arithmetic (events-maths)	Low	Check on finding	Best Practices	**Acknowledge**

Finding:

❌ LiquidityGeneratorToken.setLiquidityFeePercent(uint256) (xlight.sol:1322-1331) should emit an event for:

- _liquidityFee = liquidityFeeBps (xlight.sol#1326)

❌ LiquidityGeneratorToken.setTaxFeePercent(uint256) (xlight.sol:1314-1320) should emit an event for:

- _taxFee = taxFeeBps (xlight.sol#1315)

Recommendation:

Emit an event for critical parameter changes.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Alleviation:

X-light Team has Acknowledge this issue.

SEC-04: Reentrancy vulnerabilities leading to out-of-order Events (reentrancy-events)

Vulnerability Detail	Severity	Location	Category	Status
Reentrancy vulnerabilities leading to out-of-order Events (reentrancy-events)	Low	Check on finding	Best Practices	Acknowledge

Finding:

```
❌ Reentrancy in LiquidityGeneratorToken._transfer(address,address,uint256) (xlight.sol:1523-1561):
   • swapAndLiquify(contractTokenBalance) (xlight.sol#1548)
   • uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (xlight.sol#1609-1616)
   •
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,addre
ss(this),block.timestamp) (xlight.sol#1595-1601)
   • swapAndLiquify(contractTokenBalance) (xlight.sol#1548)
   • uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (xlight.sol#1609-1616)
   • Transfer(_msgSender(),_charityAddress,tCharity) (xlight.sol#1464)
   • _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
   • Transfer(sender,recipient,tTransferAmount) (xlight.sol#1662)
   • _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
   • Transfer(sender,recipient,tTransferAmount) (xlight.sol#1708)
   • _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
   • Transfer(sender,recipient,tTransferAmount) (xlight.sol#1685)
   • _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
   • Transfer(sender,recipient,tTransferAmount) (xlight.sol#1303)
   • _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
```

Recommendation:

Apply the [`check-effects-interactions` pattern](http://solidity.readthedocs.io/en/v0.4.21/security-considerations.html#re-entrancy).

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Alleviation:

X-light Team has Acknowledge this issue.

SEC-05: Benign reentrancy vulnerabilities (reentrancy-benign)

Vulnerability Detail	Severity	Location	Category	Status
Benign reentrancy vulnerabilities (reentrancy-benign)	Low	Check on finding	Best Practices	Acknowledge

Finding:

Reentrancy in LiquidityGeneratorToken._transfer(address,address,uint256) (xlight.sol:1523-1561):
- swapAndLiquify(contractTokenBalance) (xlight.sol#1548)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (xlight.sol#1609-1616)
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (xlight.sol#1595-1601)
- swapAndLiquify(contractTokenBalance) (xlight.sol#1548)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (xlight.sol#1609-1616)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _charityFee = _previousCharityFee (xlight.sol#1504)
- _charityFee = 0 (xlight.sol#1498)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _liquidityFee = _previousLiquidityFee (xlight.sol#1503)
- _liquidityFee = 0 (xlight.sol#1497)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _previousCharityFee = _charityFee (xlight.sol#1494)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _previousLiquidityFee = _liquidityFee (xlight.sol#1493)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _previousTaxFee = _taxFee (xlight.sol#1492)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _tFeeTotal = _tFeeTotal.add(tFee) (xlight.sol#1343)
- _tokenTransfer(from,to,amount,takeFee) (xlight.sol#1560)
- _taxFee = _previousTaxFee (xlight.sol#1502)
- _taxFee = 0 (xlight.sol#1496)

Recommendation:

Apply the [`check-effects-interactions` pattern](http://solidity.readthedocs.io/en/v0.4.21/security-considerations.html#re-entrancy).

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Alleviation:

X-light Team has Acknowledge this issue.

SEC-06: Avoid using block timestamp

Vulnerability Detail	Severity	Location	Category	Status
Avoid using block timestamp	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:

1600: block.timestamp

1615: block.timestamp

Recommendation:

Using block timestamp in smart contracts can lead to security vulnerabilities and should be avoided.

Alleviation:

X-light Team has Acknowledge this issue.

GAS-01: Use Custom Errors

Vulnerability Detail	Severity	Location	Category	Status
Avoid using block timestamp	Gas-optimization	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
169:    require(owner() == _msgSender(), "Ownable: caller is not the owner");

609:    require(isContract(target), "Address: call to non-contract");

647:    require(isContract(target), "Address: static call to non-contract");

682:    require(isContract(target), "Address: delegate call to non-contract");

1056:     require(taxFeeBps_ >= 0, "Invalid tax fee");

1057:     require(liquidityFeeBps_ >= 0, "Invalid liquidity fee");

1058:     require(charityFeeBps_ >= 0, "Invalid charity fee");

1236:     require(tAmount <= _tTotal, "Amount must be less than supply");

1261:     require(!_isExcluded[account], "Account is already excluded");

1270:     require(_isExcluded[account], "Account is already excluded");

1516:     require(owner != address(0), "ERC20: approve from the zero address");

1517:     require(spender != address(0), "ERC20: approve to the zero address");

1528:     require(from != address(0), "ERC20: transfer from the zero address");

1529:     require(to != address(0), "ERC20: transfer to the zero address");

1530:     require(amount > 0, "Transfer amount must be greater than zero");
```

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:

X-light Team 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		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External !		NO !
└	balanceOf	External !		NO !
└	transfer	External !	🛑	NO !
└	allowance	External !		NO !
└	approve	External !	🛑	NO !
└	transferFrom	External !	🛑	NO !
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└		Public !	🛑	NO !
└	owner	Public !		NO !
└	renounceOwnership	Public !	🛑	onlyOwner
└	transferOwnership	Public !	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
SafeMath	Library			
└	tryAdd	Internal 🔒		

Contract	Type	Bases		
└	trySub	Internal 🔒		
└	tryMul	Internal 🔒		
└	tryDiv	Internal 🔒		
└	tryMod	Internal 🔒		
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	sub	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	

Contract	Type	Bases		
└	functionDelegateCall	Internal 🔒	🛑	
└	verifyCallResult	Internal 🔒		
IUniswapV2Router01	Interface			
└	factory	External ❗		NO ❗
└	WETH	External ❗		NO ❗
└	addLiquidity	External ❗	🛑	NO ❗
└	addLiquidityETH	External ❗	💵	NO ❗
└	removeLiquidity	External ❗	🛑	NO ❗
└	removeLiquidityETH	External ❗	🛑	NO ❗
└	removeLiquidityWithPermit	External ❗	🛑	NO ❗
└	removeLiquidityETHWithPermit	External ❗	🛑	NO ❗
└	swapExactTokensForTokens	External ❗	🛑	NO ❗
└	swapTokensForExactTokens	External ❗	🛑	NO ❗
└	swapExactETHForTokens	External ❗	💵	NO ❗
└	swapTokensForExactETH	External ❗	🛑	NO ❗
└	swapExactTokensForETH	External ❗	🛑	NO ❗
└	swapETHForExactTokens	External ❗	💵	NO ❗
└	quote	External ❗		NO ❗
└	getAmountOut	External ❗		NO ❗
└	getAmountIn	External ❗		NO ❗
└	getAmountsOut	External ❗		NO ❗
└	getAmountsIn	External ❗		NO ❗

Contract	Type	Bases		
IUniswapV2Router02	Interface	IUniswapV2Router01		
└	removeLiquidityETHSupportingFeeOnTransferTokens	External !	🛑	NO !
└	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External !	🛑	NO !
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External !	🛑	NO !
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External !	💵	NO !
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External !	🛑	NO !
IUniswapV2Factory	Interface			
└	feeTo	External !		NO !
└	feeToSetter	External !		NO !
└	getPair	External !		NO !
└	allPairs	External !		NO !
└	allPairsLength	External !		NO !
└	createPair	External !	🛑	NO !
└	setFeeTo	External !	🛑	NO !
└	setFeeToSetter	External !	🛑	NO !
BaseToken	Implementation			
LiquidityGeneratorToken	Implementation	IERC20, Ownable, BaseToken		

Contract	Type	Bases		
└		Public ❗	💵	NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	isExcludedFromReward	Public ❗		NO❗
└	totalFees	Public ❗		NO❗
└	deliver	Public ❗	🛑	NO❗
└	reflectionFromToken	Public ❗		NO❗
└	tokenFromReflection	Public ❗		NO❗
└	excludeFromReward	Public ❗	🛑	onlyOwner
└	includeInReward	External ❗	🛑	onlyOwner
└	_transferBothExcluded	Private 🔐	🛑	
└	excludeFromFee	Public ❗	🛑	onlyOwner

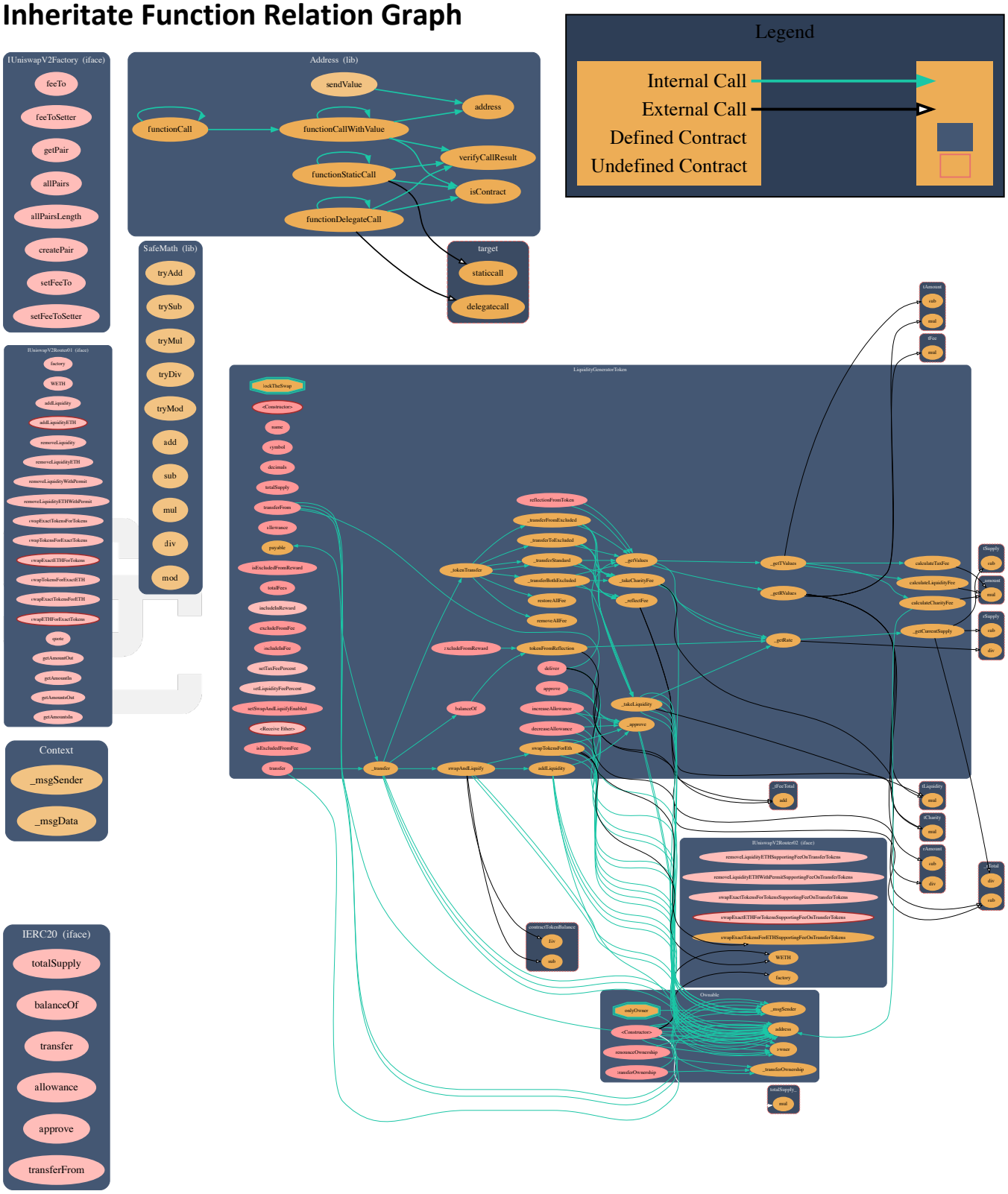
Contract	Type	Bases		
└	includeInFee	Public !		onlyOwner
└	setTaxFeePercent	External !		onlyOwner
└	setLiquidityFeePercent	External !		onlyOwner
└	setSwapAndLiquifyEnabled	Public !		onlyOwner
└		External !		NO !
└	_reflectFee	Private		
└	_getValues	Private		
└	_getTValues	Private		
└	_getRValues	Private		
└	_getRate	Private		
└	_getCurrentSupply	Private		
└	_takeLiquidity	Private		
└	_takeCharityFee	Private		
└	calculateTaxFee	Private		
└	calculateLiquidityFee	Private		
└	calculateCharityFee	Private		
└	removeAllFee	Private		
└	restoreAllFee	Private		
└	isExcludedFromFee	Public !		NO !
└	_approve	Private		
└	_transfer	Private		

Contract	Type	Bases		
└	swapAndLiquify	Private		lockTheSwap
└	swapTokensForEth	Private		
└	addLiquidity	Private		
└	_tokenTransfer	Private		
└	_transferStandard	Private		
└	_transferToExcluded	Private		
└	_transferFromExcluded	Private		

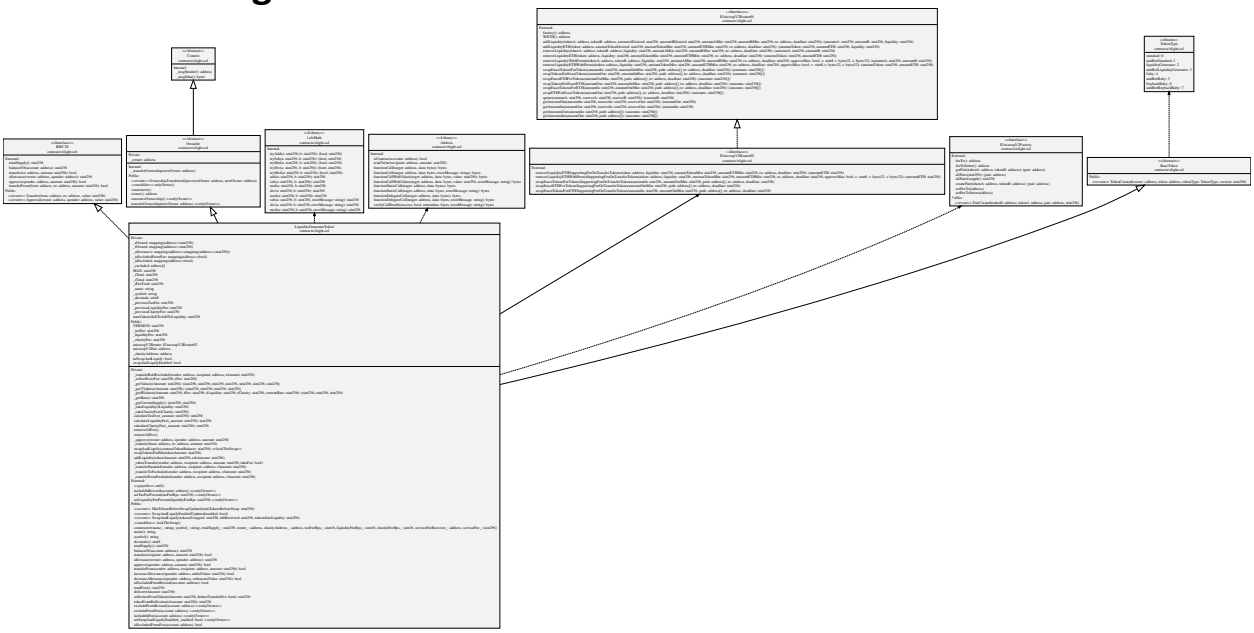
Legend

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://securi-lab.com/
Twitter	https://twitter.com/SECURI_LAB
Telegram	https://t.me/securi_lab
Medium	https://medium.com/@securi