SCRL

Full Audit Report

Hater Coin Security Re-Assessment

Hater Coin Security Re-Assessment

FULL AUDIT REPORT

Security Assessment by SCRL on **Sunday, September 17, 2023**

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

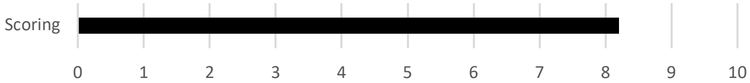
Executive Summary

For this security assessment, SCRL received a request on Sunday, September 17, 2023

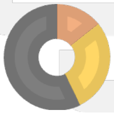
Client	Language	Audit Method	Confidential	Network Chain	Contract
Hater Coin	**Solidity**	**Whitebox**	**Public**	**Ethereum**	**0x08E1EA4e889F47D1975e09fF430Bc4C57EC16993**

Report Version	Twitter	Telegram	Website
1.2	**https://twitter.com/hatercoinmoon**	**https://t.me/RealHatercoin**	**https://hater-coin.com/**

CVSS Scoring:



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.
▪ 0	**High**			High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.
▪ 1	**Medium**	**1 Unresolved**		It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
▪ 2	**Low**	**2 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.
▪ 4	**Informational**	**4 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
▪ 5	**Gas-optimization**	**5 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
src/HATER.sol	eb3a7876a85b32cb81405fcc5e2c0de4c65603cf

Audit Version History:

Version	Date	Description
1.0	Friday, September 1, 2023	Preliminary Report
1.1	Saturday, September 2, 2023	Full Audit Report
1.2	Monday, September 18, 2023	Re-Assessment with new token contract

Audit information:

Request Date	Audit Date	Re-assessment Date
Friday, 1 September R 2023	Friday, September 1, 2023	Sunday, September 17, 2023

Smart Contract Audit Summary

SCRL

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 September 18, 2023

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: 2023.09.18 14:43:01
+07'00'

Disclaimer

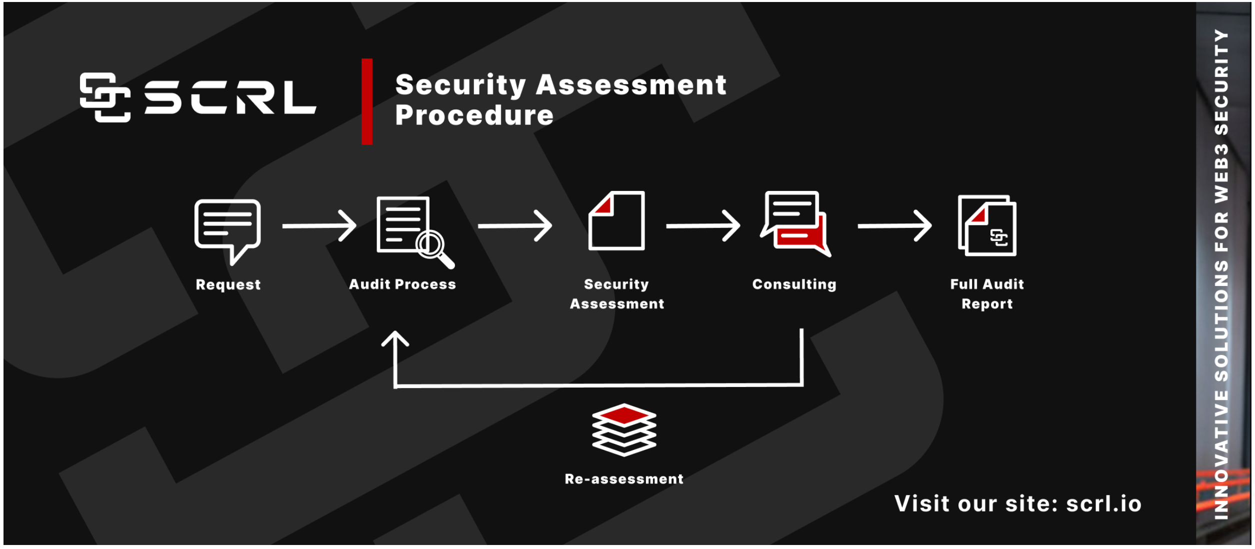
Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as "**Source code**".
And **SCRL** hereinafter referred to as "**Service Provider**", the **Service Provider** will not be held liable for any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

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 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/Hatercoin.sol	2		161	150	117	9	67	
📝 📚	**Totals**	**2**		**161**	**150**	**117**	**9**	**67**	

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	0	5	2

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
80	5

External	Internal	Private	Pure	View
66	46	4	11	30

StateVariables

Total	🌐Public
8	2

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.9`		`yes`		

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

♻️ TryCatch	Σ Unchecked

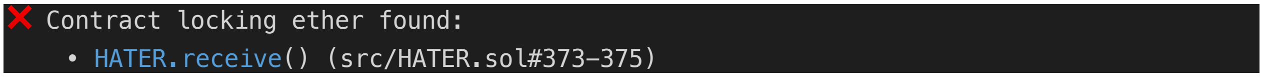
Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
SEC-01	Contracts that lock ether (locked-ether)	Medium	Best Practices	Acknowledge
SEC-02	Local variables shadowing (shadowing-local)	Low	Best Practices	Acknowledge
SEC-03	Initializers could be front-run	Low	Logical Issue	Acknowledge
SEC-04	int/uint values except 0, 1, 2, 1000 and 1e18 (pess-magic-number)	Informational	Best Practices	Acknowledge
SEC-05	Conformance to numeric notation best practices (too-many-digits)	Informational	Best Practices	Acknowledge
SEC-06	Unlocked pragma	Informational	Best Practices	Acknowledge
SEC-07	Missing checks for `address(0)` when assigning values to address state variables	Informational	Best Practices	Acknowledge
GAS-01	Use Custom Errors	Gas-optimization	Gas Optimization	Acknowledge
GAS-02	Use assembly to check for `address(0)`	Gas-optimization	Gas Optimization	Acknowledge
GAS-03	Long revert strings	Gas-optimization	Gas Optimization	Acknowledge
GAS-04	Inefficient state variable increment	Gas-optimization	Gas Optimization	Acknowledge
GAS-05	Non payable constructor	Gas-optimization	Gas Optimization	Acknowledge

SEC-01: Contracts that lock ether (locked-ether)

Vulnerability Detail	Severity	Location	Category	Status
Contracts that lock ether (locked-ether)	Medium	Check on finding	Best Practices	**Acknowledge**

Finding:



```
❌ Contract locking ether found:
    • HATER.receive() (src/HATER.sol#373-375)
```

Scenario:

Please do not send ETH or funds directly to the contract address as it may not be possible to recover the ETH. Please trade through a DEX/CEX that provides liquidity or a presale contract.

Recommendation:

Recommendation: Remove the payable attribute or add a withdraw function.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

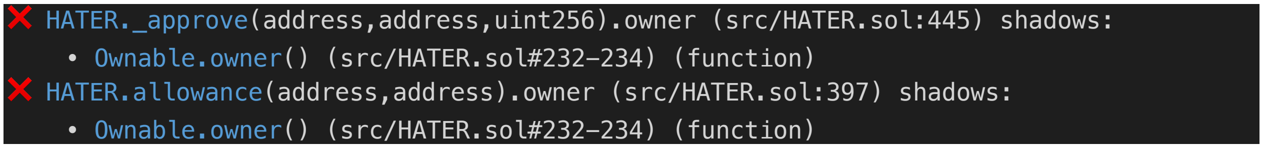
Alleviation:

Hater Coin team has acknowledge this issue.

SEC-02: Local variables shadowing (shadowing-local)

Vulnerability Detail	Severity	Location	Category	Status
Local variables shadowing (shadowing-local)	Low	Check on finding	Best Practices	Acknowledge

Finding:



```
❌ HATER._approve(address,address,uint256).owner (src/HATER.sol:445) shadows:
    • Ownable.owner() (src/HATER.sol#232-234) (function)
❌ HATER.allowance(address,address).owner (src/HATER.sol:397) shadows:
    • Ownable.owner() (src/HATER.sol#232-234) (function)
```

Recommendation:

Rename the local variables that shadow another component.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Alleviation:

Hater Coin team has acknowledge this issue.

SEC-03: Initializers could be front-run

Vulnerability Detail	Severity	Location	Category	Status
Initializers could be front-run	Low	Check on finding	Best Practices	**Acknowledge**

Finding:

```
File: HATER.sol

204:     function initialize(address, address) external;
```

Recommendation:

Initializers could be front-run, allowing an attacker to either set their own values, take ownership of the contract, and in the best case forcing a re-deployment

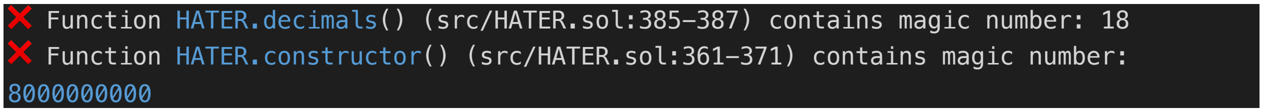
Alleviation:

Hater Coin team has acknowledge this issue.

SEC-04: int/uint values except 0, 1, 2, 1000 and 1e18 (pess-magic-number)

Vulnerability Detail	Severity	Location	Category	Status
int/uint values except 0, 1, 2, 1000 and 1e18 (pess-magic-number)	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:



```
❌ Function HATER.decimals() (src/HATER.sol:385-387) contains magic number: 18
❌ Function HATER.constructor() (src/HATER.sol:361-371) contains magic number:
8000000000
```

Recommendation:

Don't use values without assigning them to variables.

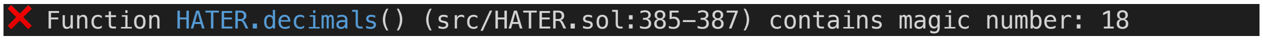
Alleviation:

Hater Coin team has acknowledge this issue.

SEC-05: 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:



```
❌ Function HATER.decimals() (src/HATER.sol:385-387) contains magic number: 18
```

Recommendation:

Don't use values without assigning them to variables.

Alleviation:

Hater Coin team has acknowledge this issue.

SEC-06: Unlocked pragma

Vulnerability Detail	Severity	Location	Category	Status
Unlocked pragma	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:

```
File: HATER.sol

3: pragma solidity ^0.8.9;
```

Recommendation:

Consider locking the compiler version to prevent unexpected behavior.

Alleviation:

Hater Coin team has acknowledge this issue.

SEC-07: Missing checks for `address(0)` when assigning values to address state variables

Vulnerability Detail	Severity	Location	Category	Status
Missing checks for `address(0)` when assigning values to address state variables	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:

```
File: HATER.sol

266:          _owner = newOwner;

368:          uniswapV2Pair = _uniswapV2Pair;

```
```

Recommendation:

when assigning values to address state variables in Solidity can lead to vulnerabilities in your smart contract. It's crucial to validate and handle the case where an address parameter is set to the zero address (address(0)), which usually represents an uninitialized or invalid address.

Alleviation:

Hater Coin team 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:

~~~
File: HATER.sol

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

260:          require(newOwner != address(0), "Ownable: new owner is the zero
address");

413:          require(currentAllowance >= subtractedValue, "HATER: decreased allowance
below zero");

426:          require(currentAllowance >= amount, "HATER: transfer amount exceeds
allowance");

436:          require(sender != address(0), "HATER: transfer from the zero address");

437:          require(recipient != address(0), "HATER: transfer to the zero address");

439:          require(senderBalance >= amount, "HATER: transfer amount exceeds
balance");

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

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

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

```
~~~

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/)

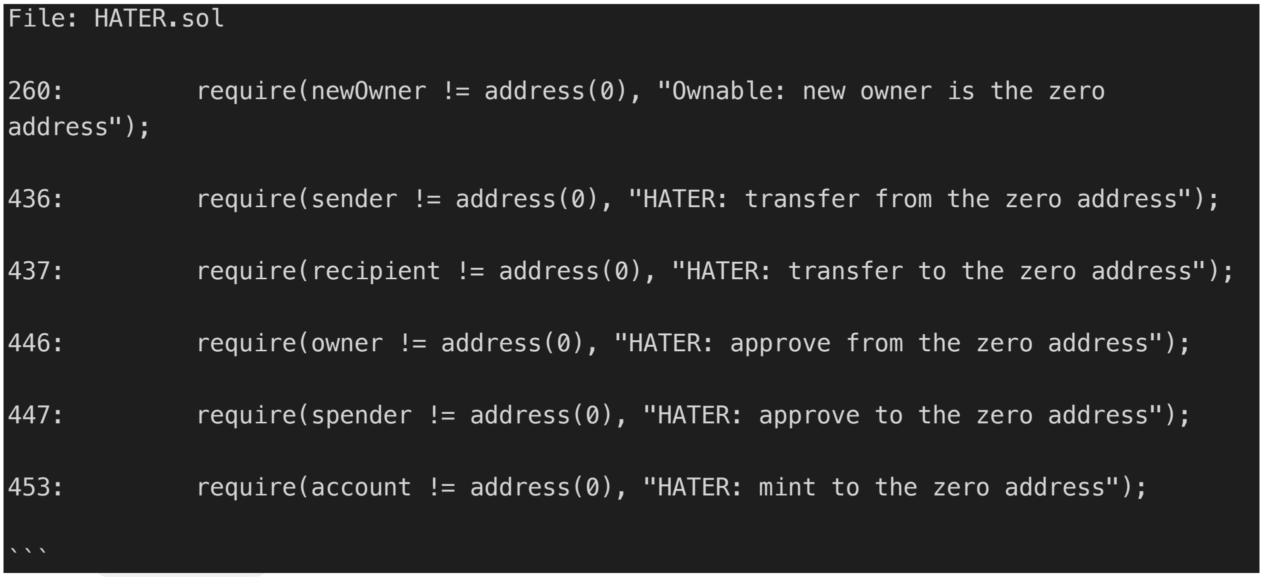
Alleviation:

Hater Coin team has acknowledge this issue.

GAS-02: Use assembly to check for `address(0)`

Vulnerability Detail	Severity	Location	Category	Status
Use assembly to check for `address(0)`	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:



```
File: HATER.sol

260:         require(newOwner != address(0), "Ownable: new owner is the zero
address");

436:         require(sender != address(0), "HATER: transfer from the zero address");

437:         require(recipient != address(0), "HATER: transfer to the zero address");

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

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

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

```
```

Alleviation:

Hater Coin team has acknowledge this issue.
```

## GAS-03: Long revert strings

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Long revert strings | - | Check on finding | Gas Optimization | **Acknowledge** |

## Finding:

```
File: HATER.sol

260:         require(newOwner != address(0), "Ownable: new owner is the zero
address");

413:         require(currentAllowance >= subtractedValue, "HATER: decreased allowance
below zero");

426:         require(currentAllowance >= amount, "HATER: transfer amount exceeds
allowance");

436:         require(sender != address(0), "HATER: transfer from the zero address");

437:         require(recipient != address(0), "HATER: transfer to the zero address");

439:         require(senderBalance >= amount, "HATER: transfer amount exceeds
balance");

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

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

```
```

Alleviation:

Hater Coin team has acknowledge this issue.
```

## GAS-04: Inefficient state variable increment

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Inefficient state variable increment | - | Check on finding | Gas Optimization | **Acknowledge** |

## Finding:

```
File: HATER.sol

454:         _totalSupply += amount;
```

Recommendation:

<x> += <y> costs more gas than <x> = <x> + <y> for state variables.

References:

https://gist.github.com/IllIllI000/cbbfb267425b898e5be734d4008d4fe8

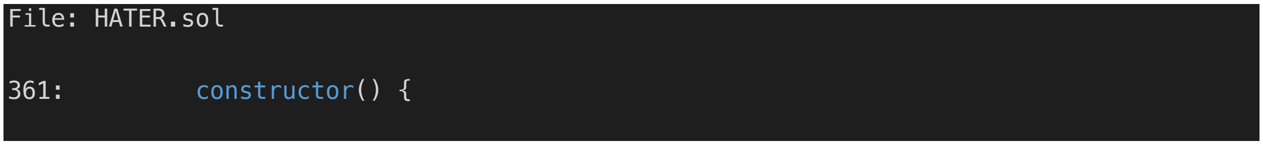
Alleviation:

Hater Coin team has acknowledge this issue.

GAS-05: Non payable constructor

Vulnerability Detail	Severity	Location	Category	Status
Non payable constructor	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:



```
File: HATER.sol

361:        constructor() {
```

Recommendation:

Consider making costructor payable to save gas.

References:

https://twitter.com/0xAsm0d3us/status/1518960704271056897

Alleviation:

Hater Coin 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**		
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IUniswapV2Router01	Interface			
L	factory	External ❗		NO ❗
L	WETH	External ❗		NO ❗
L	addLiquidity	External ❗	🛑	NO ❗
L	addLiquidityETH	External ❗	💵	NO ❗
L	removeLiquidity	External ❗	🛑	NO ❗
L	removeLiquidityETH	External ❗	🛑	NO ❗
L	removeLiquidityWithPermit	External ❗	🛑	NO ❗
L	removeLiquidityETHWithPermit	External ❗	🛑	NO ❗
L	swapExactTokensForTokens	External ❗	🛑	NO ❗
L	swapTokensForExactTokens	External ❗	🛑	NO ❗
L	swapExactETHForTokens	External ❗	💵	NO ❗
L	swapTokensForExactETH	External ❗	🛑	NO ❗
L	swapExactTokensForETH	External ❗	🛑	NO ❗
L	swapETHForExactTokens	External ❗	💵	NO ❗
L	quote	External ❗		NO ❗
L	getAmountOut	External ❗		NO ❗
L	getAmountIn	External ❗		NO ❗

Contract	Type	Bases		
L	getAmountsOut	External ❗		NO ❗
L	getAmountsIn	External ❗		NO ❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
L	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
L	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO ❗
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
IUniswapV2Factory	Interface			
L	feeTo	External ❗		NO ❗
L	feeToSetter	External ❗		NO ❗
L	getPair	External ❗		NO ❗
L	allPairs	External ❗		NO ❗
L	allPairsLength	External ❗		NO ❗
L	createPair	External ❗	🛑	NO ❗
L	setFeeTo	External ❗	🛑	NO ❗
L	setFeeToSetter	External ❗	🛑	NO ❗

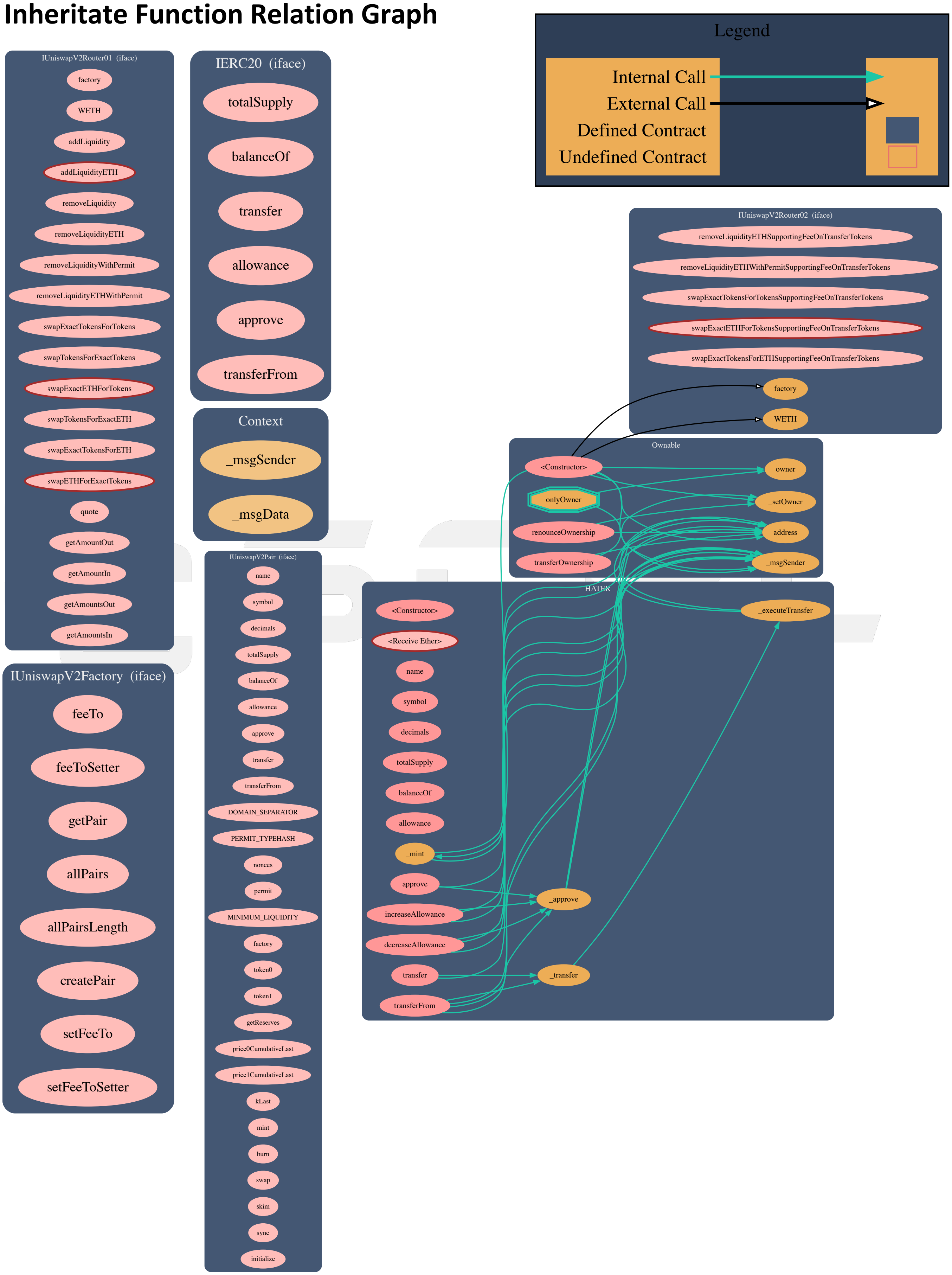
Contract	Type	Bases		
IUniswapV2Pair	Interface			
L	name	External ❗		NO ❗
L	symbol	External ❗		NO ❗
L	decimals	External ❗		NO ❗
L	totalSupply	External ❗		NO ❗
L	balanceOf	External ❗		NO ❗
L	allowance	External ❗		NO ❗
L	approve	External ❗	🛑	NO ❗
L	transfer	External ❗	🛑	NO ❗
L	transferFrom	External ❗	🛑	NO ❗
L	DOMAIN_SEPARATOR	External ❗		NO ❗
L	PERMIT_TYPEHASH	External ❗		NO ❗
L	nonces	External ❗		NO ❗
L	permit	External ❗	🛑	NO ❗
L	MINIMUM_LIQUIDITY	External ❗		NO ❗
L	factory	External ❗		NO ❗
L	token0	External ❗		NO ❗
L	token1	External ❗		NO ❗
L	getReserves	External ❗		NO ❗
L	price0CumulativeLast	External ❗		NO ❗
L	price1CumulativeLast	External ❗		NO ❗

Contract	**Type**	**Bases**		
L	kLast	External ❗		NO ❗
L	mint	External ❗	🛑	NO ❗
L	burn	External ❗	🛑	NO ❗
L	swap	External ❗	🛑	NO ❗
L	skim	External ❗	🛑	NO ❗
L	sync	External ❗	🛑	NO ❗
L	initialize	External ❗	🛑	NO ❗
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
Ownable	Implementation	Context		
L		Public ❗	🛑	NO ❗
L	owner	Public ❗		NO ❗
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
L	_setOwner	Private 🔐	🛑	
IERC20	Interface			
L	totalSupply	External ❗		NO ❗
L	balanceOf	External ❗		NO ❗

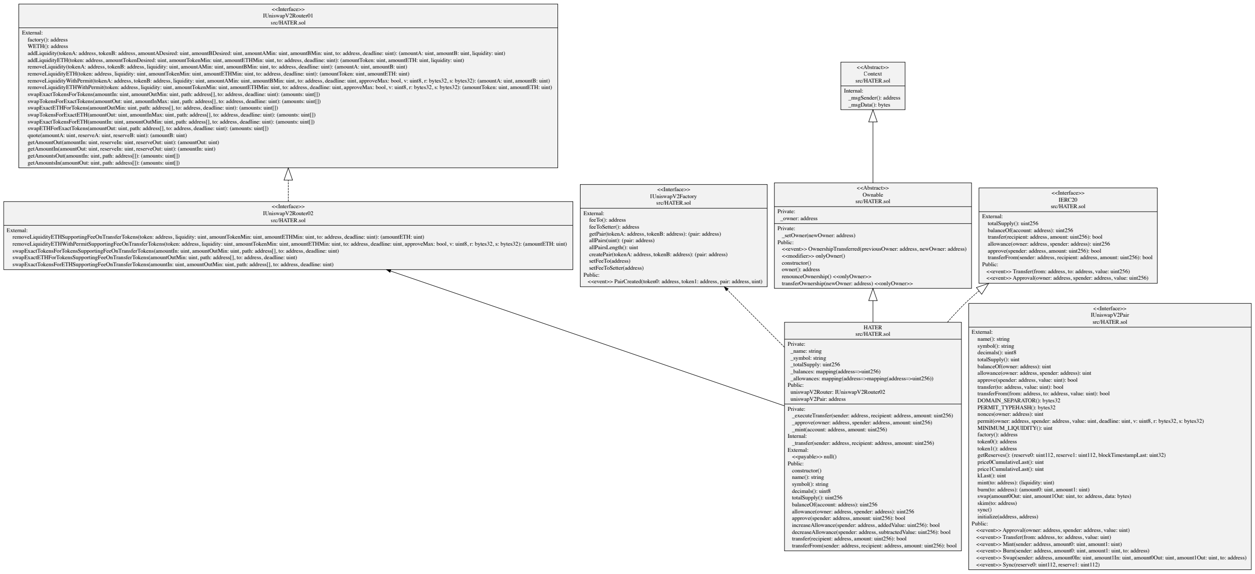
Contract	**Type**	**Bases**		
L	transfer	External ❗	🛑	NO ❗
L	allowance	External ❗		NO ❗
L	approve	External ❗	🛑	NO ❗
L	transferFrom	External ❗	🛑	NO ❗
HATER	Implementation	Ownable, IERC20		
L		Public ❗	🛑	NO ❗
L		External ❗	💵	NO ❗
L	name	Public ❗		NO ❗
L	symbol	Public ❗		NO ❗
L	decimals	Public ❗		NO ❗
L	totalSupply	Public ❗		NO ❗
L	balanceOf	Public ❗		NO ❗
L	allowance	Public ❗		NO ❗
L	approve	Public ❗	🛑	NO ❗
L	increaseAllowance	Public ❗	🛑	NO ❗
L	decreaseAllowance	Public ❗	🛑	NO ❗
L	transfer	Public ❗	🛑	NO ❗
L	transferFrom	Public ❗	🛑	NO ❗
L	_transfer	Internal 🔒	🛑	
L	_executeTransfer	Private 🔐	🛑	

Contract	Type	Bases		
L	_approve	Private 🔐	🛑	
L	_mint	Private 🔐	🛑	

Inheritate Function Relation Graph



UML Class Diagram


<<Interface>>
IUniswapV2Router01
src/HATER.sol
External:
factory(): address
WETH(): address
addLiquidity(tokenA: address, tokenB: address, amountADesired: uint, amountBDesired: uint, amountAMin: uint, amountBMin: uint, to: address, deadline: uint): (amountA: uint, amountB: uint, liquidity: uint)
addLiquidityETH(token: address, amountTokenDesired: uint, amountTokenMin: uint, amountETHMin: uint, to: address, deadline: uint): (amountToken: uint, amountETH: uint, liquidity: uint)
removeLiquidity(tokenA: address, tokenB: address, liquidity: uint, amountAMin: uint, amountBMin: uint, to: address, deadline: uint): (amountA: uint, amountB: uint)
removeLiquidityETH(token: address, liquidity: uint, amountTokenMin: uint, amountETHMin: uint, to: address, deadline: uint): (amountToken: uint, amountETH: uint)
removeLiquidityWithPermit(tokenA: address, tokenB: address, liquidity: uint, amountAMin: uint, amountBMin: uint, to: address, deadline: uint, approveMax: bool, v: uint8, r: bytes32, s: bytes32): (amountA: uint, amountB: uint)
removeLiquidityETHWithPermit(token: address, liquidity: uint, amountTokenMin: uint, amountETHMin: uint, to: address, deadline: uint, approveMax: bool, v: uint8, r: bytes32, s: bytes32): (amountToken: uint, amountETH: uint)
swapExactTokensForTokens(amountIn: uint, amountOutMin: uint, path: address[], to: address, deadline: uint): (amounts: uint[])
swapTokensForExactTokens(amountOut: uint, amountInMax: uint, path: address[], to: address, deadline: uint): (amounts: uint[])
swapExactETHForTokens(amountOutMin: uint, path: address[], to: address, deadline: uint): (amounts: uint[])
swapTokensForExactETH(amountOut: uint, amountInMax: uint, path: address[], to: address, deadline: uint): (amounts: uint[])
swapExactTokensForETH(amountIn: uint, amountOutMin: uint, path: address[], to: address, deadline: uint): (amounts: uint[])
swapETHForExactTokens(amountOut: uint, path: address[], to: address, deadline: uint): (amounts: uint[])
quote(amountA: uint, reserveA: uint, reserveB: uint): (amountB: uint)
getAmountOut(amountIn: uint, reserveIn: uint, reserveOut: uint): (amountOut: uint)
getAmountIn(amountOut: uint, reserveIn: uint, reserveOut: uint): (amountIn: uint)
getAmountsOut(amountIn: uint, path: address[]): (amounts: uint[])
getAmountsIn(amountOut: uint, path: address[]): (amounts: uint[])
<<Abstract>>
Context
src/HATER.sol
Internal:
_msgSender(): address
_msgData(): bytes
<<Interface>>
IUniswapV2Router02
src/HATER.sol
External:
removeLiquidityETHSupportingFeeOnTransferTokens(token: address, liquidity: uint, amountTokenMin: uint, amountETHMin: uint, to: address, deadline: uint): (amountETH: uint)
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(token: address, liquidity: uint, amountTokenMin: uint, amountETHMin: uint, to: address, deadline: uint, approveMax: bool, v: uint8, r: bytes32, s: bytes32): (amountETH: uint)
swapExactTokensForTokensSupportingFeeOnTransferTokens(amountIn: uint, amountOutMin: uint, path: address[], to: address, deadline: uint)
swapExactETHForTokensSupportingFeeOnTransferTokens(amountOutMin: uint, path: address[], to: address, deadline: uint)
swapExactTokensForETHSupportingFeeOnTransferTokens(amountIn: uint, amountOutMin: uint, path: address[], to: address, deadline: uint)
<<Interface>>
IUniswapV2Factory
src/HATER.sol
External:
feeTo(): address
feeToSetter(): address
getPair(tokenA: address, tokenB: address): (pair: address)
allPairs(uint): (pair: address)
allPairsLength(): uint
createPair(tokenA: address, tokenB: address): (pair: address)
setFeeTo(address)
setFeeToSetter(address)
Public:
<<event>> PairCreated(token0: address, token1: address, pair: address, uint)
<<Abstract>>
Ownable
src/HATER.sol
Private:
_owner: address
Private:
_setOwner(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Interface>>
IERC20
src/HATER.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)
HATER
src/HATER.sol
Private:
_name: string
_symbol: string
_totalSupply: uint256
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
Public:
uniswapV2Router: IUniswapV2Router02
uniswapV2Pair: address
Private:
_executeTransfer(sender: address, recipient: address, amount: uint256)
_approve(owner: address, spender: address, amount: uint256)
_mint(account: address, amount: uint256)
Internal:
_transfer(sender: address, recipient: address, amount: uint256)
External:
<<payable>> null()
Public:
constructor()
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
increaseAllowance(spender: address, addedValue: uint256): bool
decreaseAllowance(spender: address, subtractedValue: uint256): bool
transfer(recipient: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
<<Interface>>
IUniswapV2Pair
src/HATER.sol
External:
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint
balanceOf(owner: address): uint
allowance(owner: address, spender: address): uint
approve(spender: address, value: uint): bool
transfer(to: address, value: uint): bool
transferFrom(from: address, to: address, value: uint): bool
DOMAIN_SEPARATOR(): bytes32
PERMIT_TYPEHASH(): bytes32
nonces(owner: address): uint
permit(owner: address, spender: address, value: uint, deadline: uint, v: uint8, r: bytes32, s: bytes32)
MINIMUM_LIQUIDITY(): uint
factory(): address
token0(): address
token1(): address
getReserves(): (reserve0: uint112, reserve1: uint112, blockTimestampLast: uint32)
price0CumulativeLast(): uint
price1CumulativeLast(): uint
kLast(): uint
mint(to: address): (liquidity: uint)
burn(to: address): (amount0: uint, amount1: uint)
swap(amount0Out: uint, amount1Out: uint, to: address, data: bytes)
skim(to: address)
sync()
initialize(address, address)
Public:
<<event>> Approval(owner: address, spender: address, value: uint)
<<event>> Transfer(from: address, to: address, value: uint)
<<event>> Mint(sender: address, amount0: uint, amount1: uint)
<<event>> Burn(sender: address, amount0: uint, amount1: uint, to: address)
<<event>> Swap(sender: address, amount0In: uint, amount1In: uint, amount0Out: uint, amount1Out: uint, to: address)
<<event>> Sync(reserve0: uint112, reserve1: uint112)

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/