SECURI LAB

Full Audit Report

Dogens Launchpad Security Assessment

Real Cybersecurity
Protecting digital assets

SECURI LAB

Made in Thailand

SECURI LAB
Audit Report

SECURI LAB
(THAILAND) contact@securi-lab.com

FULL AUDIT REPORT

Table of Contents 1

Report Information

About Report	**Dogens Launchpad Security Assessment**
Version	**v1.1**
Client	**Dogens**
Language	**Solidity**
Confidentiality	**Public**
Contract File	**mastergenPad.sol** SHA-1: 43bc66b786e22a3429bd6184ba56272680832d62 **This audit uses the file as the client submitted. Please check with a differential checker after the smart contract code has been deployed and verified.**
Audit Method	**Whitebox**
Security Assessment Author	**Auditor** **Mark K. [Security Researcher | Redteam]** **Kevin N. [Security Researcher | Web3 Dev]** **Yusheng T. [Security Researcher | Incident Response]** **Approve Document** **Ronny C. CTO & Head of Security Researcher** **Chinnakit J. CEO & Founder**

*Audit Method

Whitebox: SECURI LAB Team receives all source code from the client to provide the assessment.
Blackbox: SECURI LAB Team receives only bytecode from the client to provide the assessment.

Digital Sign (Only Full Audit Report)

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: 2023.07.18 20:35:41 +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 **SECURI Lab** 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 Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.

SECURI LAB disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull

The SECURI LAB team has conducted a comprehensive security assessment of the vulnerabilities.
This assessment is tested with an expert assessment. Using the following test requirements

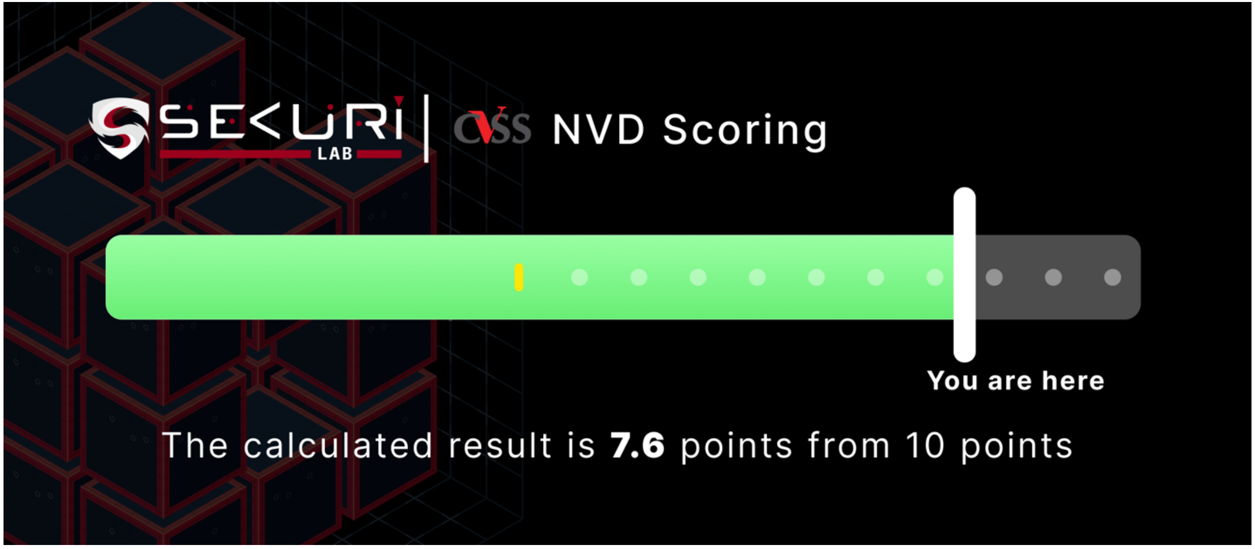
1. Smart Contract Testing with Expert Analysis By testing the most common and uncommon vulnerabilities.
2. Automated program testing It includes a sample vulnerability test and a sample of the potential vulnerabilities being used for the most frequent attacks.
3. Manual Testing with AST/WAS/ASE/SMT and reviewed code line by line
4. Visibility, Mutability, Modifier function testing, such as whether a function can be seen in general, or whether a function can be changed and if so, who can change it.
5. Function association test It will be displayed through the association graph.
6. This safety assessment is cross-checked prior to the delivery of the assessment results.

Executive Summary

For this security assessment, SECURI LAB received a request from Genpad on Thursday, April 20, 2023.

NVD CVSS Scoring

The score was calculated using the NVD (National Vulnerability Database) of NIST (National Institute of Standards and Technology) under the CVSS 3.1 standard, based on the CIA (Confidentiality, Integrity, Availability).

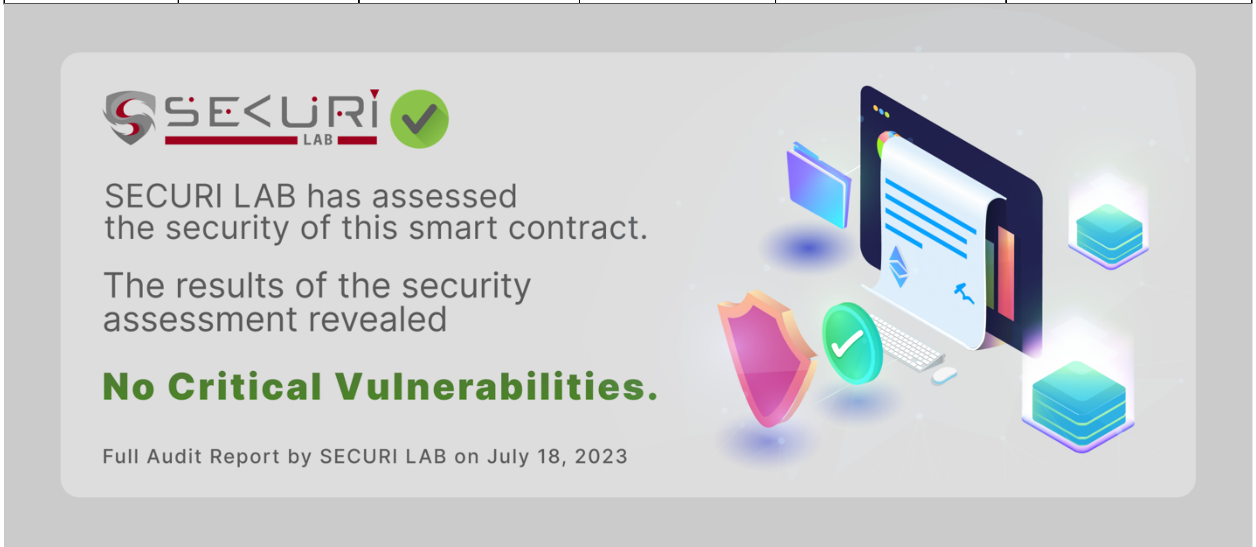


The calculated result is **7.6** points from 10 points

Audit Result

SECURI LAB evaluated the smart contract security of the project and found: **[Total : 14]**

Critical	High	Medium	Low	Very Low	Informational
0	1	0	2	0	2



SECURI LAB has assessed the security of this smart contract.

The results of the security assessment revealed

No Critical Vulnerabilities.

Full Audit Report by SECURI LAB on July 18, 2023

Project Introduction

Scope Information:

Project Name	Dogens Launchpad
Website	https://dogens.io/
Chain	Ethereum Mainnet
Language	Solidity

Audit Information:

Request Date	Thursday, April 20, 2023
Audit Date	Friday, April 21, 2023
Re-assessment Date	-

Audit Version History:

Version	Date	Description
1.0	Sunday, April 23, 2023	Preliminary Report
1.1	Tuesday, July 18, 2023	Full Audit Report

Initial Audit Scope:

Smart Contract File	**mastergenPad.sol** SHA-1: 43bc66b786e22a3429bd6184ba56272680832d62 **This audit uses the file as the client submitted. Please check with a differential checker after the smart contract code has been deployed and verified.**
Compiler Version	**v0.8.17**

Source Units in Scope

Source Units Analyzed: 3
Source Units in Scope: 3 (**100%**)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/mastergenPad.sol	1		273	273	188	41	175	💰📤
📝	**Totals**	**1**		**273**	**273**	**188**	**41**	**175**	💰📤

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, ...)

Dependencies / External Imports

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	1
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	1

Description Report Files Description Table

File Name	SHA-1 Hash
contracts/mastergenPad.sol	43bc66b786e22a3429bd6184ba56272680832d62

Security Assessment Procedure

Securi has the following procedures and regulations for conducting security assessments:

1.Request Audit Client submits a form request through the Securi channel. After receiving the request, Securi will discuss a security assessment. And drafting a contract and agreeing to sign a contract together with the Client

2.Auditing Securi performs security assessments of smart contracts obtained through automated analysis and expert manual audits.

3.Preliminary Report At this stage, Securi will deliver an initial security assessment. To report on vulnerabilities and errors found under Audit Scope will not publish preliminary reports for safety.

4.Reassessment After Securi has delivered the Preliminary Report to the Client, Securi will track the status of the vulnerability or error, which will be published to the Final Report at a later date with the following statuses:

- **a.Acknowledge** The client has been informed about errors or vulnerabilities from the security assessment.
- **b.Resolved** The client has resolved the error or vulnerability. Resolved is probably just a commit, and Securi is unable to verify that the resolved has been implemented or not.
- **c.Decline** Client has rejected the results of the security assessment on the issue.

5.Final Report Securi providing full security assessment report and public

Request Audit
Auditing
Preliminary Report
Reassessment
Final Report

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

Both have a total of 3 levels: **High**, **Medium**, **Low**. By *Informational* will not be classified as a level

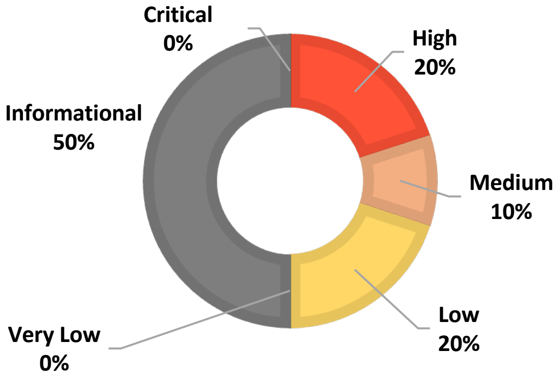
Confidence Impact [Likelihood]	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

Vulnerability Severity Summary

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
5 categories based on the **lowest severity**:
Very Low, Low, Medium, **High**, Critical.
For **Informational** & will **Non-class/Optimization/Best-practices will** <u>not be counted</u> as **severity**



Vulnerability Severity Level	Total
Critical	0
High	1
Medium	0
Low	2
Very Low	0
Informational	2
Non-class/Optimization/Best-practices	9

Category information:

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.

FULL AUDIT REPORT

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
SEC-01	Centralization Risk	High	Centralization	Acknowledge
SEC-02	Avoid using block timestamp	Low	Best Practices	Acknowledge
SEC-03	Empty Function Body - Consider commenting why	Low	Coding Style	Acknowledge
SEC-04	Avoid using tx.origin	Informational	Best Practices	Acknowledge
SEC-05	unlocked-compiler-version	Informational	Best Practices	Acknowledge
NC-01	Functions not used internally could be marked external	-	Gas Optimization	Acknowledge
GAS-01	Use `selfbalance()` instead of `address(this).balance`	-	Gas Optimization	Acknowledge
GAS-02	Use assembly to check for `address(0)`	-	Gas Optimization	Acknowledge
GAS-03	Using bools for storage incurs overhead	-	Gas Optimization	Acknowledge
GAS-04	Cache array length outside of loop	-	Gas Optimization	Acknowledge
GAS-05	Use calldata instead of memory for function arguments that do not get mutated	-	Gas Optimization	Acknowledge
GAS-06	Use shift Right/Left instead of division/multiplication if possible	-	Gas Optimization	Acknowledge
GAS-07	Use Custom Errors	-	Gas Optimization	Acknowledge
GAS-08	Use != 0 instead of > 0 for unsigned integer comparison	-	Gas Optimization	Acknowledge

SEC-01: Centralization Risk

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

Finding:

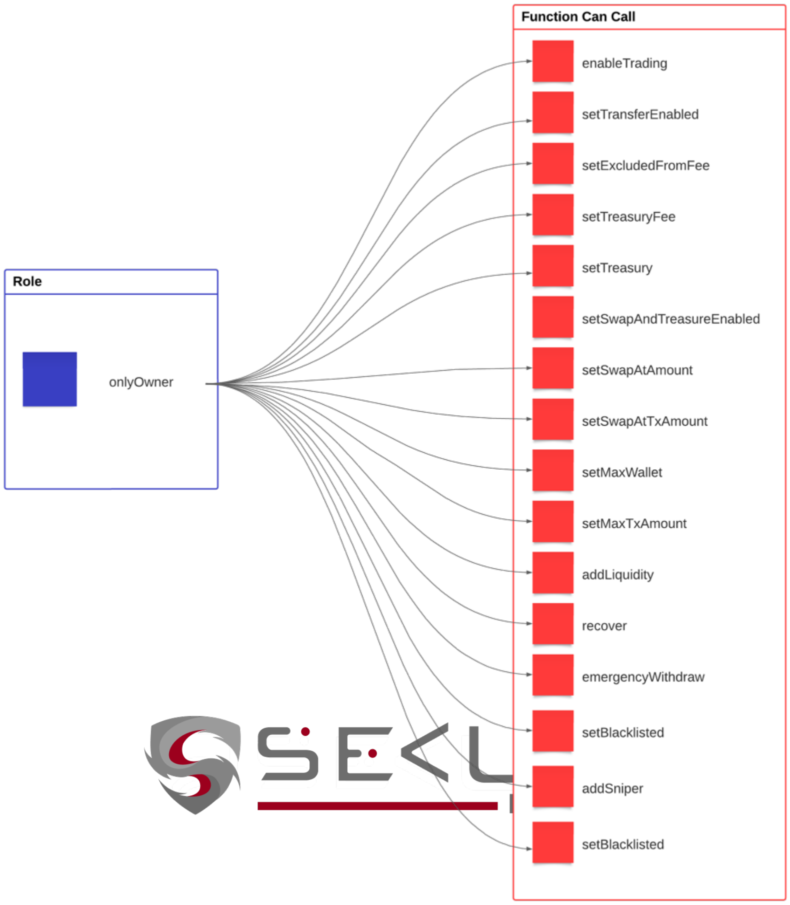
```
9: contract TokenTest is ERC20, Ownable {

159:     function enableTrading() external onlyOwner {

166:     function setTransferEnabled(bool _state) external onlyOwner {

171:     function setExcludedFromFee(address _account, bool _state) external onlyOwner
{

176:     function setTreasuryFee(uint256 _feeOnBuy, uint256 _feeOnSell) external
onlyOwner {

182:     function setTreasury(address payable _treasuryWallet) external onlyOwner {

186:     function setSwapAndTreasureEnabled(bool _state) external onlyOwner {

190:     function setSwapAtAmount(uint256 _amount) external onlyOwner {

194:     function setSwapAtTxAmount(uint256 _amount) external onlyOwner {

198:     function setMaxWallet(uint256 _amount) external onlyOwner {

202:     function setMaxTxAmount(uint256 _amount) external onlyOwner {

207:     function addLiquidity(uint256 _tokenAmount) external payable onlyOwner {

212:     function recover(address _token, uint256 _amount) external onlyOwner {

240:     function emergencyWithdraw() external onlyOwner {

244:     function setBlacklisted(address _account, bool _state) external onlyOwner {

253:     function addSniper(address[] memory account) external onlyOwner {

264:     function setBlacklisted(address[] memory _accounts, bool[] memory _states)
external onlyOwner {
```

Scenario:

Centralized risk refers to the potential security risks that arise when a smart contract is controlled by a central entity or a single point of failure. If the contract is controlled by a central authority, then the contract may be vulnerable to attacks that target the centralized entity.
Centralized risk that can lead to rug pulls typically arises from the centralization of control or ownership of a project's assets, particularly in decentralize d finance (DeFi) projects built on blockchain platforms like Ethereum.

FULL AUDIT REPORT

Contract TokenTest (File: mastergenPad.sol)



In the TokenTest (File: mastergenPad.sol) contract, Owner can call functions enableTrading, setTransferEnabled, setExcludedFromFee, setTreasuryFee, setTreasury, setSwapAndTreasureEnabled, setSwapAtAmount, setSwapAtTxAmount, setMaxWallet, setMaxTxAmount, addLiquidity, recover, emergencyWithdraw, setBlacklisted, addSniper, setBlacklisted. We've found that some functions work in an anti-whale manner and allow the owner to pause trading. Assigning a backlist address and also another function we recommend that for transparency use Timelock to increase the delay for users. Function calls are visible before they are fully executed. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

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.

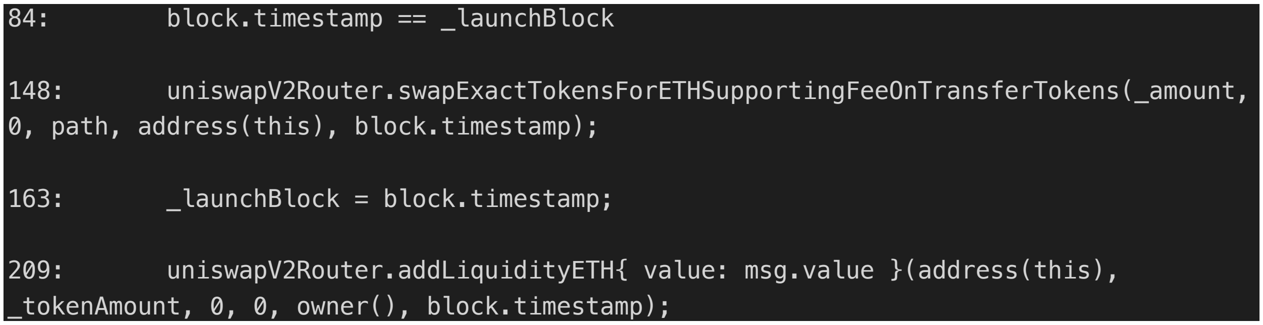
Alleviation:

Dogens Team has acknowledge this issue.

SEC-02: Avoid using block timestamp

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

Finding:



```
84:        block.timestamp == _launchBlock

148:       uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(_amount,
0, path, address(this), block.timestamp);

163:       _launchBlock = block.timestamp;

209:       uniswapV2Router.addLiquidityETH{ value: msg.value }(address(this),
_tokenAmount, 0, 0, owner(), block.timestamp);
```

Recommendation:

Using block timestamp in smart contracts can lead to security vulnerabilities and should be avoided.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Exploit Scenario:

-

Alleviation:

Dogens Team has acknowledge this issue.

SEC-03: Empty Function Body - Consider commenting why

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

Finding:

```
214:                IERC20(_token).transfer(msg.sender, _amount);
```

Recommendation:

While this line of code might work correctly in most cases, it is considered potentially unsafe because it assumes the ERC20 token contract implements the transfer function correctly and consistently with the ERC20 standard.

A safer way to interact with ERC20 tokens is to use a widely-adopted and audited library, such as OpenZeppelin's ERC20 library. This helps you avoid potential issues with non-standard or malicious token implementations.

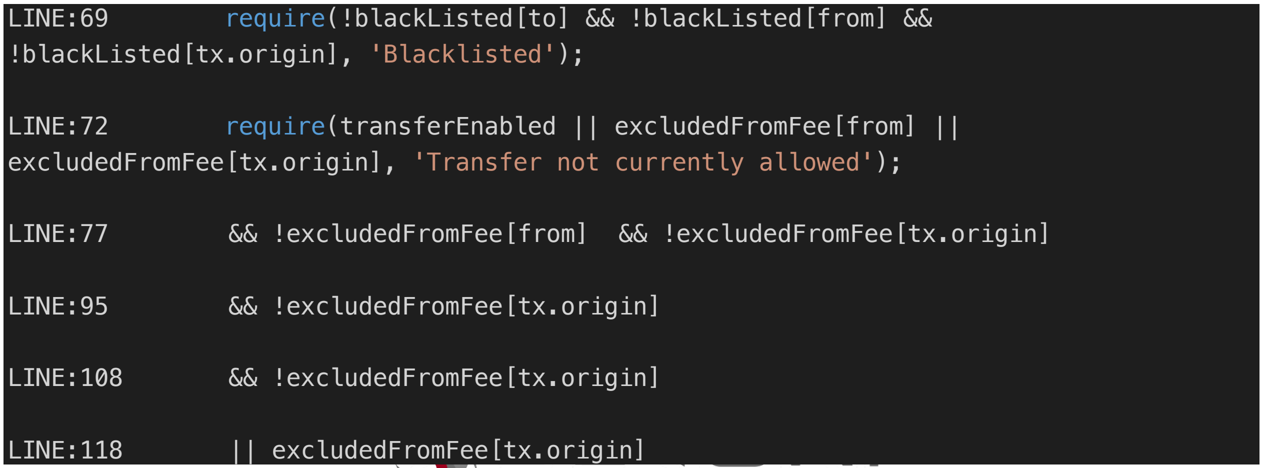
Alleviation:

Dogens Team has acknowledge this issue.

SEC-04: Avoid using tx.origin

Vulnerability Detail	Severity	Location	Category	Status
Avoid using tx.origin	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:



```
LINE:69        require(!blackListed[to] && !blackListed[from] &&
!blackListed[tx.origin], 'Blacklisted');

LINE:72        require(transferEnabled || excludedFromFee[from] ||
excludedFromFee[tx.origin], 'Transfer not currently allowed');

LINE:77        && !excludedFromFee[from]  && !excludedFromFee[tx.origin]

LINE:95        && !excludedFromFee[tx.origin]

LINE:108       && !excludedFromFee[tx.origin]

LINE:118       || excludedFromFee[tx.origin]
```

Recommendation:

Using tx.origin in smart contracts can expose them to potential vulnerabilities, particularly in cases where the contract relies on tx.origin for authentication or authorization. The primary concern with using tx.origin is that it can be susceptible to phishing attacks when used to determine the sender of a transaction. Instead of using tx.origin, it is recommended to use msg.sender. The msg.sender variable represents the direct caller of the current function, whereas tx.origin represents the originator of the entire transaction call chain.

Exploit Scenario:

-

Alleviation:

Dogens Team has acknowledge this issue.

SEC-05: unlocked-compiler-version

Vulnerability Detail	Severity	Location	Category	Status
unlocked-compiler-version	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:

```
2: pragma solidity ^0.8.17;
```

Recommendation:

Unlocked pragma disables all source code analysis, making it vulnerable to attacks

Exploit Scenario:

-

Alleviation:

Dogens Team has acknowledge this issue.

NC-01: Functions not used internally could be marked external

Vulnerability Detail	Severity	Location	Category	Status
Functions not used internally could be marked external	-	Check on finding	Best Practices	**Acknowledge**

Finding:

```
57:      function decimals() public view override returns (uint8) {
```

Recommendation:

is marked **public**, which means it can be called both internally (from within the contract) and externally (from outside the contract). However, if this function is not intended to be used internally, it can be marked **external** to optimize gas usage and restrict its usage to external calls only.

Exploit Scenario:

-

Alleviation:

Dogens Team has acknowledge this issue.

GAS-01: Use `selfbalance()` instead of `address(this).balance`

Vulnerability Detail	Severity	Location	Category	Status
Use `selfbalance()` instead of `address(this).balance`	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
150:        uint256 ethBalance = address(this).balance;

241:        payable(owner()).call{ value: address(this).balance }('');
```

Recommendation:

Use assembly when getting a contract's balance of ETH.

You can use `selfbalance()` instead of `address(this).balance` when getting your contract's balance of ETH to save gas.
Additionally, you can use `balance(address)` instead of `address.balance()` when getting an external contract's balance of ETH.

Saves 15 gas when checking internal balance, 6 for external

Alleviation:

Dogens 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:

```
65:          require(to != address(0), 'Transfer to zero address');

213:          if (_token != address(0)) {
```

Recommendation:

Saves 6 gas per instance

Instances (2):

Alleviation:

Dogens Team has acknowledge this issue.

GAS-03: Using bools for storage incurs overhead

Vulnerability Detail	Severity	Location	Category	Status
Using bools for storage incurs overhead	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
13:     bool private _inSwapAndLiquify;

16:     bool public tradingEnabled;

17:     bool public transferEnabled;

18:     bool public swapAndTreasureEnabled;

20:     mapping(address => bool) public blackListed;

21:     mapping(address => bool) public excludedFromFee;
```

Recommendation:

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See [source] (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27).

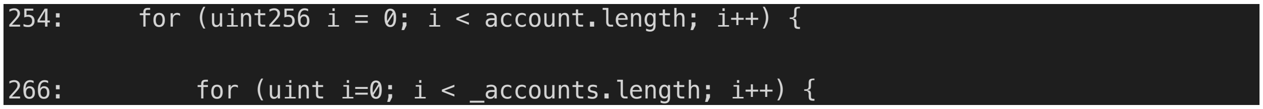
Alleviation:

Dogens Team has acknowledge this issue.

GAS-04: Cache array length outside of loop

Vulnerability Detail	Severity	Location	Category	Status
Cache array length outside of loop	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:



```
254:     for (uint256 i = 0; i < account.length; i++) {

266:         for (uint i=0; i < _accounts.length; i++) {
```

Recommendation:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Alleviation:

Dogens Team has acknowledge this issue.

GAS-05: Use calldata instead of memory for function arguments that do not get mutated

Vulnerability Detail	Severity	Location	Category	Status
Use calldata instead of memory for function arguments that do not get mutated	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
264:     function setBlacklisted(address[] memory _accounts, bool[] memory _states) 
external onlyOwner {

264:     function setBlacklisted(address[] memory _accounts, bool[] memory _states) 
external onlyOwner {
```

Recommendation:

Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.

Alleviation:

Dogens Team has acknowledge this issue.

GAS-06: Use shift Right/Left instead of division/multiplication if possible

Vulnerability Detail	Severity	Location	Category	Status
Use shift Right/Left instead of division/multiplication if possible	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
52:         swapAtTxAmount = totalSupply() / 1000; // 0.1%

52:         swapAtTxAmount = totalSupply() / 1000; // 0.1%
```

Recommendation:

Shifting left by N is like multiplying by 2^N and shifting right by N is like dividing by 2^N

Alleviation:

Dogens Team has acknowledge this issue.

GAS-07: Use Custom Errors

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

Finding:

```
217:                require(success, "Can't send ETH");

265:             require(_accounts.length == _states.length && _accounts.length > 0,
"wrong input");
```

Recommendation:

[Source](https://blog.soliditylang.org/2021/04/21/custom-errors/)
Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Alleviation:

Dogens Team has acknowledge this issue.

GAS-08: Use != 0 instead of > 0 for unsigned integer comparison

Vulnerability Detail	Severity	Location	Category	Status
Use != 0 instead of > 0 for unsigned integer comparison	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
265:          require(_accounts.length == _states.length && _accounts.length > 0,
"wrong input");
```

Recommendation:

[Source](https://blog.soliditylang.org/2021/04/21/custom-errors/)
Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Alleviation:

Dogens 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

FULL AUDIT REPORT

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

Visibility, Mutability, Modifier function testing

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
1	0	0	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
17	2

External	Internal	Private	Pure	View
16	18	0	0	2

StateVariables

Total	🌐Public
18	14

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.17`		`yes`		

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

FULL AUDIT REPORT

♻ TryCatch	Σ Unchecked

Contracts Description Table

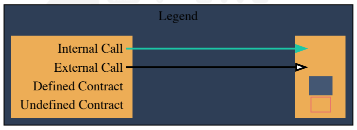
Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
TokenTest	Implementation	ERC20, Ownable		
└		Public ❗	🛑	ERC20
└	decimals	Public ❗		NO ❗
└	_transfer	Internal 🔒	🛑	
└	_swapAndSendTreasure	Internal 🔒	🛑	lockTheSwap
└	enableTrading	External ❗	🛑	onlyOwner
└	setTransferEnabled	External ❗	🛑	onlyOwner
└	setExcludedFromFee	External ❗	🛑	onlyOwner
└	setTreasuryFee	External ❗	🛑	onlyOwner
└	setTreasury	External ❗	🛑	onlyOwner
└	setSwapAndTreasureEnabled	External ❗	🛑	onlyOwner
└	setSwapAtAmount	External ❗	🛑	onlyOwner
└	setSwapAtTxAmount	External ❗	🛑	onlyOwner
└	setMaxWallet	External ❗	🛑	onlyOwner
└	setMaxTxAmount	External ❗	🛑	onlyOwner

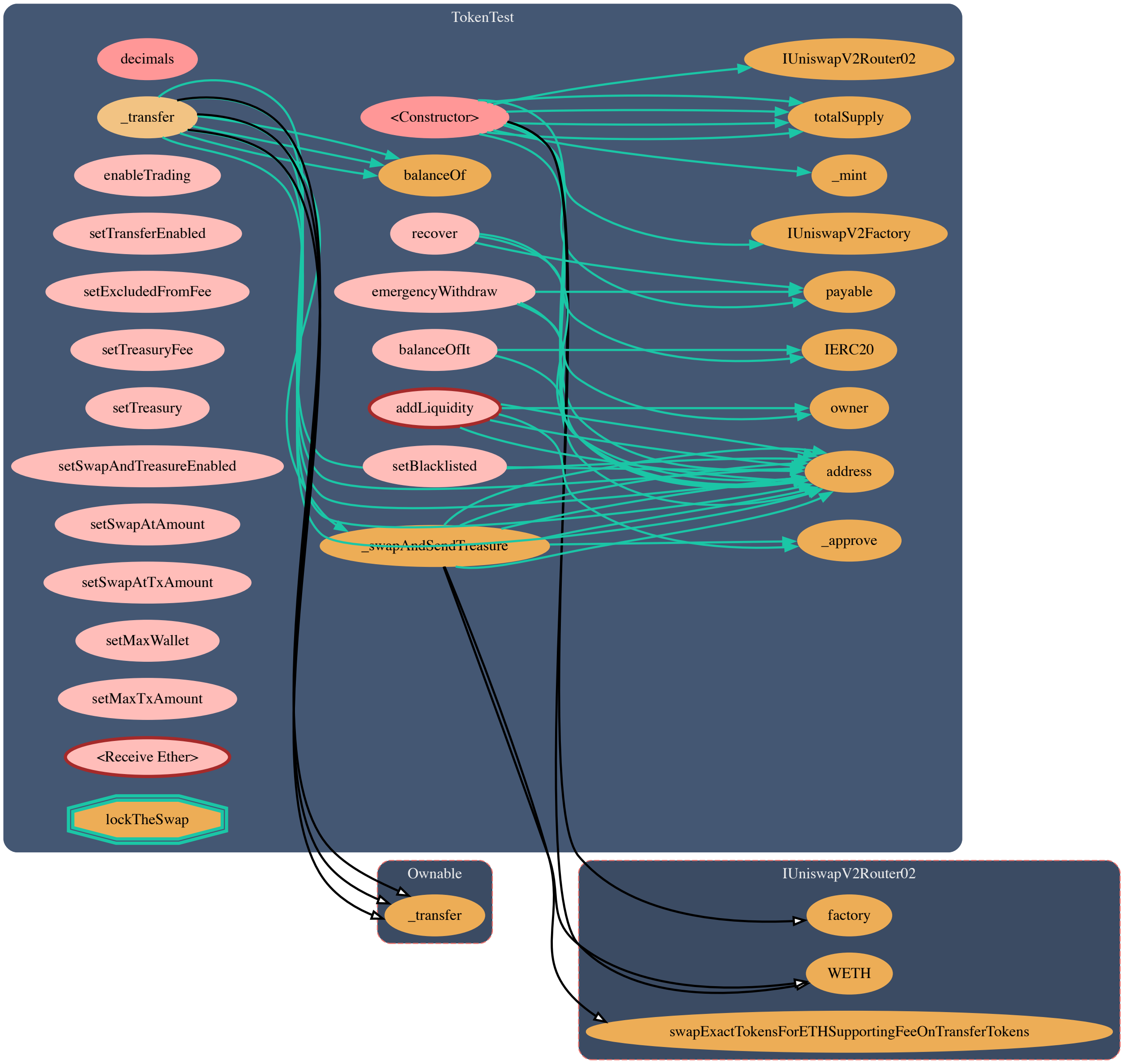
Contract	Type	Bases		
└	addLiquidity	External ❗	💵	onlyOwner
└	recover	External ❗	🛑	onlyOwner
└		External ❗	💵	NO❗
└	balanceOfIt	External ❗		NO❗
└	emergencyWithdraw	External ❗	🛑	onlyOwner
└	setBlacklisted	External ❗	🛑	onlyOwner

Legend

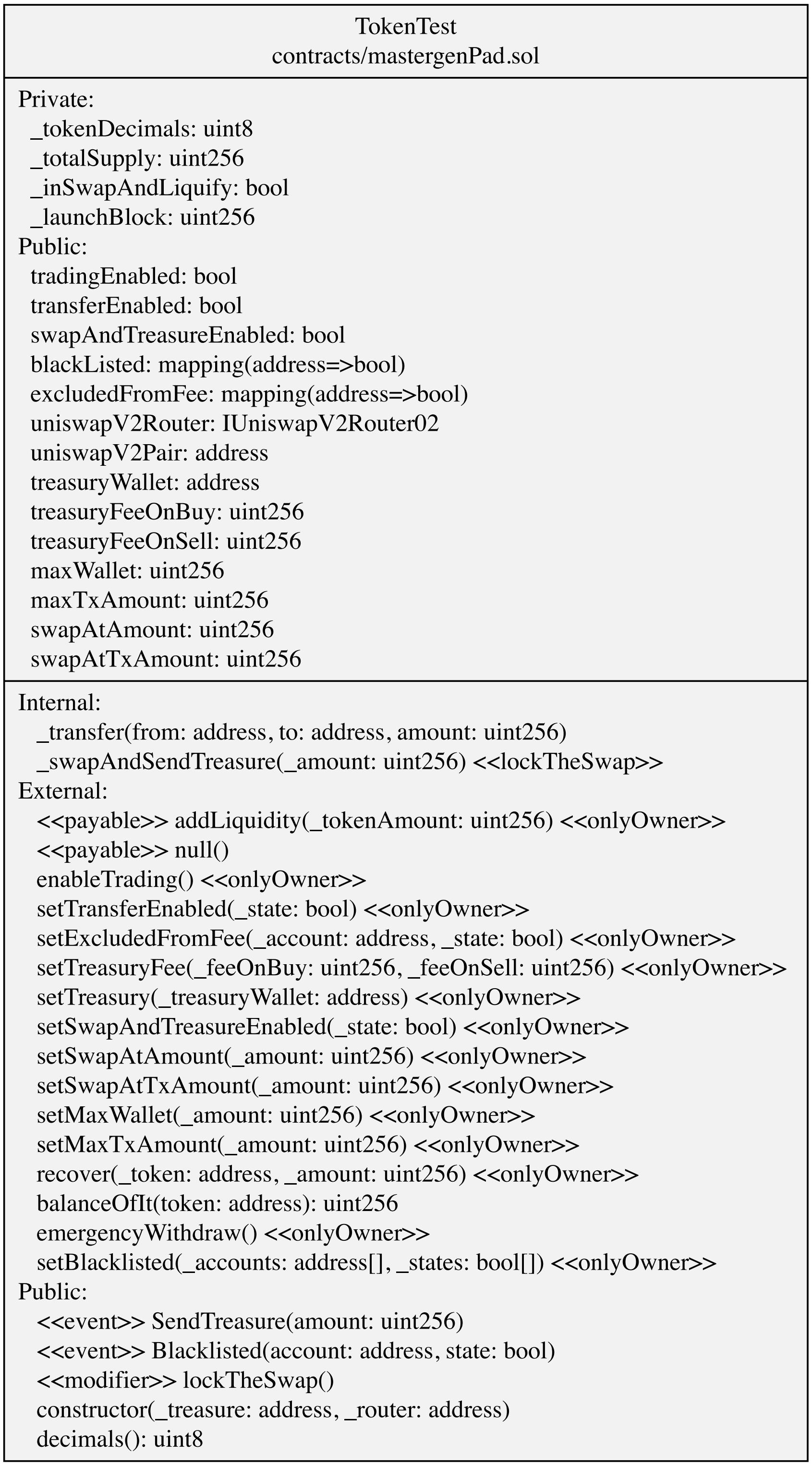
Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Inheritate Function Relation Graph





UML Class Diagram



```
TokenTest
contracts/mastergenPad.sol

Private:
  _tokenDecimals: uint8
  _totalSupply: uint256
  _inSwapAndLiquify: bool
  _launchBlock: uint256
Public:
  tradingEnabled: bool
  transferEnabled: bool
  swapAndTreasureEnabled: bool
  blackListed: mapping(address=>bool)
  excludedFromFee: mapping(address=>bool)
  uniswapV2Router: IUniswapV2Router02
  uniswapV2Pair: address
  treasuryWallet: address
  treasuryFeeOnBuy: uint256
  treasuryFeeOnSell: uint256
  maxWallet: uint256
  maxTxAmount: uint256
  swapAtAmount: uint256
  swapAtTxAmount: uint256

Internal:
   _transfer(from: address, to: address, amount: uint256)
   _swapAndSendTreasure(_amount: uint256) <<lockTheSwap>>
External:
   <<payable>> addLiquidity(_tokenAmount: uint256) <<onlyOwner>>
   <<payable>> null()
   enableTrading() <<onlyOwner>>
   setTransferEnabled(_state: bool) <<onlyOwner>>
   setExcludedFromFee(_account: address, _state: bool) <<onlyOwner>>
   setTreasuryFee(_feeOnBuy: uint256, _feeOnSell: uint256) <<onlyOwner>>
   setTreasury(_treasuryWallet: address) <<onlyOwner>>
   setSwapAndTreasureEnabled(_state: bool) <<onlyOwner>>
   setSwapAtAmount(_amount: uint256) <<onlyOwner>>
   setSwapAtTxAmount(_amount: uint256) <<onlyOwner>>
   setMaxWallet(_amount: uint256) <<onlyOwner>>
   setMaxTxAmount(_amount: uint256) <<onlyOwner>>
   recover(_token: address, _amount: uint256) <<onlyOwner>>
   balanceOfIt(token: address): uint256
   emergencyWithdraw() <<onlyOwner>>
   setBlacklisted(_accounts: address[], _states: bool[]) <<onlyOwner>>
Public:
   <<event>> SendTreasure(amount: uint256)
   <<event>> Blacklisted(account: address, state: bool)
   <<modifier>> lockTheSwap()
   constructor(_treasure: address, _router: address)
   decimals(): uint8
```

About SECURI LAB

SECURI LAB is a group of cyber security experts providing cyber security consulting, smart contract security audits, and KYC services.

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