



Full Audit Report

Degcom Security Assessment



Degcom Security Assessment

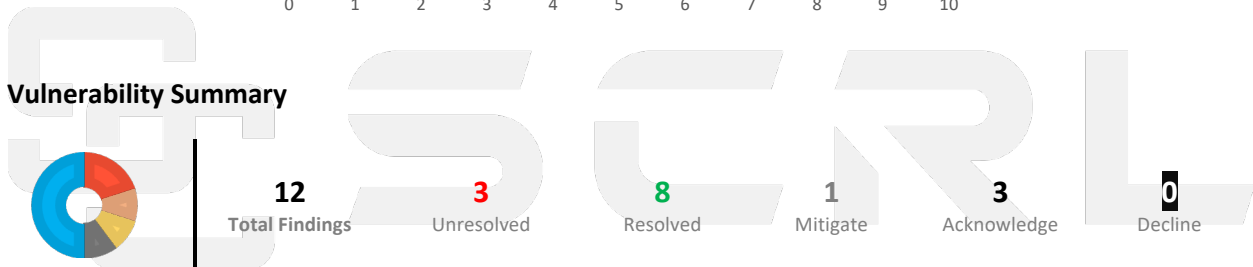
FULL AUDIT REPORTSecurity Assessment by SCRL on **Friday, November 22, 2024**

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

**This code security assessment is performed on contracts deployed on a base testnet.****Executive Summary**For this security assessment, SCRL received a request on **Friday, November 8, 2024**

Client	Language	Audit Method	Confidential	Network Chain	Contract
Degcom	Solidity	Whitebox	Public	Base Testnet	0x05D21b19aECD5cA024ee5112E6E170256795FbE4

Report Version	Twitter	Telegram	Website
1.1	https://x.com/degcom_io	https://t.me/DegComPortal	https://degcom.io/

Scoring:**Vulnerability Summary**▪ **0 Critical**

Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.

▪ **2 High****1 Resolved, 1 Partially Resolved**

High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.

▪ **1 Medium****1 Resolved**

It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.

▪ **1 Low****1 Resolved**

While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.

▪ **0 Very Low**

Very Low severity is used for minor security concerns that have minimal impact and are generally of low risk.

▪ **3 Informational****2 Resolved, 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

▪ **5 Gas-optimization****3 Resolved, 2 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/DegComBase.sol	5c58493a4d268760c1e483d07f60f4a69c3bb4f3

Audit Version History:

Version	Date	Description
1.0	Tuesday, November 12, 2024	Preliminary Report
1.1	Friday, 22 November, 2024	Full audit report [with re-assessment]

Audit information:

Request Date	Audit Date	Re-assessment Date
Friday, November 8, 2024	Tuesday, November 12, 2024	Friday, 22 November, 2024

Smart Contract Audit Summary



**SCRL has assessed
the security of this smart contract.**

**The results of the security
assessment revealed**

No Critical Vulnerabilities.


Full Audit Report by SCRL on November 22, 2024



Security Assessment Author

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

Digital Sign



ID: 46A7CFF1-4FAA-4ECB-81C8-33516EDF334C
Reason: Digitally signed by <contact@scrl.io>
November 22, 2024 11:18 AM +07

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.

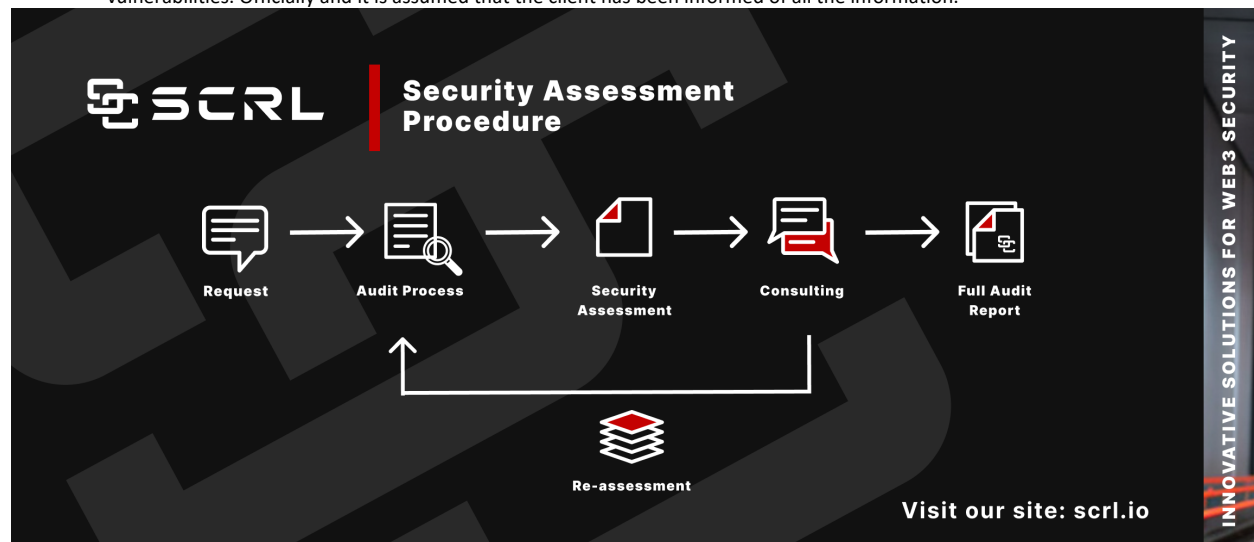
If the **service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security 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) and Formal Verification
3. **Security Assessment** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
4. **Consulting** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
 - a. **Re-assessment** Reassess the security when the client implements the source code improvements and if the client is satisfied with the results of the audit. We will proceed to the next step.
5. **Full Audit Report** SCRL provides clients with official security assessment reports informing them of risks and vulnerabilities. Officially and it is assumed that the client has been informed of all the information.



Risk Rating

Risk rating using this commonly defined: $Risk\ rating = impact * confidence$

Impact The severity and potential impact of an attacker attack
Confidence Ensuring that attackers expose and use this vulnerability

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

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

$Risk\ rating = impact * confidence$

It is categorized into

7 categories severity based



For **Informational & Non-class/Optimization/Best-practices** will not be counted as severity

Category

Centralization Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission	Economics Risk Risks that may affect the economic mechanism system, such as the ability to increase Mint token	Logical Issue Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.	Authorization Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.	Mathematical Mathematical Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.	Naming Conventions Naming Conventions naming variables that may affect code understanding or naming inconsistencies
Security Risk Security Risk of loss or damage if it's no mitigate	Coding Style Coding Style is Tips coding for efficiency performance	Best Practices Best Practices is suggestions for improvement	Optimization Optimization is performance improvement	Gas Optimization Gas Optimization is increase performance to avoid expensive gas	Dead Code Dead Code having unused code This may result in wasted resources and gas fees.

Table Of Content

Summary

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

Source Code Detail

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

Vulnerability Finding




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

About SCRL

Source Units in Scope

Source Units Analyzed: 1

Source Units in Scope: 1 (100%)

T y p e	File	Logi c Con trac ts	Interfa ces	Li ne s	nL ine s	nS LOC	Co mm ent Line s	Co mpl ex. Sco re	Capa biliti es
	contracts/DegComBase.sol	1		422	391	301	19	297	
	Totals	1		422	391	301	19	297	

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	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			
12	7			
External	Internal	Private	Pure	View
12	5	1	1	1

StateVariables

Total	 Public
6	4

Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<input type="text" value="^0.8.20"/>		<input type="text" value="yes"/>		<input type="text" value="yes"/>	
 Transfers ETH	 Low-Level Calls	 Delegate Call	 Uses Hash Functions	 ECRewriter	 New/Create/Create2
<input type="text" value="yes"/>		<input type="text" value="yes"/>			

 TryCatch	Σ Unchecked

Dependencies / External Imports

Dependency / Import Path	Count
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1
@openzeppelin/contracts/utils/ReentrancyGuard.sol	1
@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol	1
@uniswap/v3-core/contracts/libraries/FixedPoint96.sol	1



Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
BUG-01	Compile errors	BUG	-	Resolved
REN-01	Potential Reentrancy Attack	High	Logical Issue	Resolved
DLC-01	Potential Unsafe Delegate call	High	Security Risk	Partially Resolved
URV-01	Unchecked Return Value	Medium	Logical Issue	Resolved
SEC-01	Missing Zero Address Validation	Low	Best Practices	Resolved
SEC-02	Costly operations in a loop	Informational	Best Practices	Acknowledge
SEC-03	Functions not used internally could be marked external	Informational	Best Practices	Resolved
SEC-04	Return values of `approve()` not checked	Informational	Naming Conventions	Resolved
GAS-01	Cache array length outside of loop	Gas-optimization	Gas Optimization	Acknowledge
GAS-02	Use Custom Errors	Gas-optimization	Gas Optimization	Acknowledge
GAS-03	Functions guaranteed to revert when called by normal users can be marked `payable`	Gas-optimization	Gas Optimization	Resolved
GAS-04	`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)	Gas-optimization	Gas Optimization	Resolved
GAS-05	Use `!= 0` instead of `> 0` for unsigned integer comparison	Gas-optimization	Gas Optimization	Resolved

BUG-01: Compile errors

Vulnerability Detail	Severity	Location	Category	Status
Compile errors	BUG	Check on finding	-	Resolved

Finding:

```
Log Error: Compiler error Stack too deep, try removing local variables.
DegComBase.sol:263:34: CompilerError: Stack too deep, try removing local variables.
    IBEP20(path[1]).transfer(path[2], refFee);
                          ^__^
```

Description:

The contract **DegComBase.sol** is encountering a **"Stack too deep"** compiler error due to excessive local variables in the function containing line 263. This error occurs when a function attempts to use more than 16 local variables on the EVM stack.

Impact

- Contract fails to compile
- Function execution is blocked
- Core functionality of token transfers is impacted

Recommendation:

Refactor the code to use storage variables instead of local variables to avoid the stack limitation:

Example Refactor Code:

Refactor on line:

L231: IBEP20(path[0]).transferFrom(msg.sender, **_path2**, refFee);

L265: IBEP20(path[1]).transfer(**_path2**, refFee);

Add function between:

L212 - L213: **address _path2 = path[2];**

L246 - L248: **address _path2 = path[2];**

*The highlighted code has been refactored/added.

References: Solidity 0.7.6 Docs [Storage, Memory and the stack]
<https://docs.soliditylang.org/en/v0.7.6/introduction-to-smart-contracts.html?highlight=storage%20memory%20stack#storage-memory-and-the-stack>

stack exchange [error when compiling stack too deep]
<https://ethereum.stackexchange.com/questions/6061/error-while-compiling-stack-too-deep>

*****We strongly advise against using the `vialR = true` option to bypass the compiler and successfully compile Solidity code. This approach poses a significant risk of encountering errors in the IR Pipeline, as the Solidity version 0.7.6 is still in an experimental and unstable state.**

Alleviation:

The Degcom Team has already resolved the issue.



REN-01: Potential Reentrancy Attack

Vulnerability Detail	Severity	Location	Category	Status
Potential Reentrancy Attack	High	Check on finding	Logical Issue	Resolved

Finding:

```
L98:         router.exactInputSingle{value: amountAfterTax}(params);

L101:        router.swapExactETHForTokensSupportingFeeOnTransferTokens{

L111:        (bool refSentResult,) = refAddress.call{value: refFee}("");

L166:        IBEP20(path[0]).transferFrom(msg.sender, address(this), feeParams[5]);

L167:        IBEP20(path[0]).approve(dexAddress, feeParams[5]);

L170:        router.exactInputSingle(params);

L174:        weth.withdraw(finalBalance);

L190:        (bool withdrawSuccess,) = payable(msg.sender).call{value:
amountAfterFee}("");

L193:        (bool refSentResult,) = refAddress.call{value: refFee}("");

L230:        IBEP20(path[1]).transfer(msg.sender, finalBalance);

L231:        IBEP20(path[0]).transferFrom(msg.sender, _path2, refFee);

L264:        IBEP20(path[1]).transfer(msg.sender, amountAfterFee);

L265:        IBEP20(path[1]).transfer(_path2, refFee);
```

Description:

The DegComBase contract contains multiple high-risk reentrancy vectors in its token swapping and fee distribution functions. The contract handles ETH and ERC20 tokens while interacting with Uniswap V3 and custom DEX routers, creating several potential attack surfaces.

Impact

1. Direct fund loss through:
 - Token balance manipulation
 - Fee bypass
 - ETH drain
2. State corruption leading to:
 - Incorrect fee distribution
 - Balance tracking issues
 - Platform fee evasion

Recommendation:

To prevent reentrancy attacks, it is recommended to use the Checks-Effects-Interactions pattern and consider adding a reentrancy guard (nonReentrant) from OpenZeppelin's ReentrancyGuard.

Example Added OpenZeppelin Reentrancy Guard Code:

Add Library Reentrancy Guard between:

```
L5:  import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

Using Reentrancy Guard

```
L13:  contract DegComBase is ReentrancyGuard {
```

Modify Function with **nonReentrant**

```
L59:  function degComBuy(...) external payable nonReentrant {
```

```
L128: function degComSell(...) external payable nonReentrant {
```

References: SWC-107: Reentrancy: <https://swcregistry.io/docs/SWC-107>

OpenZeppelin ReentrancyGuard:
<https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard>

Alleviation:

The Degcom Team has already resolved the issue.

DLC-01: Potential Unsafe Delegate call

Vulnerability Detail	Severity	Location	Category	Status
Potential Unsafe Delegate call	High	Check on finding	Logical Issue	Partially Resolved

Finding:

```

L176:          (bool success,) =
dexAddress.delegatecall(abi.encodeWithSelector(IDEXRouter.swapExactTokensForETHSupport
ingFeeOnTransferTokens.selector, feeParams[5], feeParams[4], tokenPath, address(this),
block.timestamp + 10));

L223:          (bool success,) =
path[3].delegatecall(abi.encodeWithSelector(IDEXRouter.swapExactTokensForTokensSupport
ingFeeOnTransferTokens.selector, amountAfterTax, feeParams[4], tokenPath,
address(this), block.timestamp + 10));

L253:          (bool success,) =
path[3].delegatecall(abi.encodeWithSelector(IDEXRouter.swapExactTokensForTokensSupport
ingFeeOnTransferTokens.selector, feeParams[5], feeParams[4], tokenPath, address(this),
block.timestamp + 10));

```

Description:

The contract uses delegatecall to execute external DEX router functions without proper validation or access controls. delegatecall preserves the context of the calling contract, allowing the called contract to modify the caller's state variables and balance, creating significant security risks.

Recommendation:

Avoid using `delegatecall`. Use only trusted destinations.

References: SWC-112: <https://swcregistry.io/docs/SWC-112/>

Controlled Delegatecall destination:
<https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall>

Alleviation:

The Degcom Team has already partially resolved the issue.

22 Nov 2024

Partially resolved this issue, however, since the degcom contract requires delegatecall, the degcom team has added whitelisted address checker functionality has been added to reduced impact the delegatecall issue but it's still have delegatecall in use.

URV-01: Unchecked Return Value

Vulnerability Detail	Severity	Location	Category	Status
Unchecked Return Value	Medium	Check on finding	Logical Issue	Resolved

Finding:

```
L166:      IBEP20(path[0]).transferFrom(msg.sender, address(this), feeParams[5]);
L167:      IBEP20(path[0]).approve(dexAddress, feeParams[5]);
L230:      IBEP20(path[1]).transfer(msg.sender, finalBalance);
L231:      IBEP20(path[0]).transferFrom(msg.sender, _path2, refFee);
L232:      IBEP20(path[0]).transferFrom(msg.sender, platformWallet,
finalPlatformFee);
L264:      IBEP20(path[1]).transfer(msg.sender, amountAfterFee);
L265:      IBEP20(path[1]).transfer(_path2, refFee);
L266:      IBEP20(path[1]).transfer(platformWallet, finalPlatformFee);
```

Description

The current implementation directly calls transfer/transferFrom functions without checking their return values. Per the ERC20 standard, these functions return a boolean indicating success or failure. Some token implementations, most notably USDT, return false instead of reverting on failure. Without checking these return values, the contract might continue execution even when transfers fail.

Recommendation:

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

References: OpenZeppelin SafeERC20 Documentation:
<https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20>

Alleviation:

The Degcom Team has already resolved the issue.

SEC-01: Missing Zero Address Validation

Vulnerability Detail	Severity	Location	Category	Status
Missing Zero Address Validation	Low	Check on finding	Best Practices	Resolved

Finding:

```
DegComBase.degComBuy(address[],uint256[]).refAddress (src/DegComBase.sol:80) lacks
a zero-check on :
    • (refSentResult,None) = refAddress.call{value: refFee}()
(src/DegComBase.sol#111)
DegComBase.degComBuyToken(address[],uint256[]) (src/DegComBase.sol:202-235) uses
timestamp for comparisons
    • require(bool,string)(success,Delegate Call failed) (src/DegComBase.sol#224)
DegComBase.degComSell(address[],uint256[]) (src/DegComBase.sol:128-200) uses
timestamp for comparisons
    • require(bool,string)(success,Delegate Call failed) (src/DegComBase.sol#179)
DegComBase.degComSell(address[],uint256[]).refAddress (src/DegComBase.sol:144) lacks
a zero-check on :
    • (refSentResult,None) = refAddress.call{value: refFee}()
(src/DegComBase.sol#193)
```

Recommendation:

Check that the address is not zero.

References: Missing Zero Address Validation - Slither
<https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

Alleviation:

The Degcom Team has already resolved the issue.

SEC-02: Costly operations in a loop

Vulnerability Detail	Severity	Location	Category	Status
Costly operations in a loop	Informational	Check on finding	Best Practices	Acknowledge

Finding:

DegComBase.removeFromWhitelist(address[]) (src/DegComBase.sol:297-303) has costly operations inside a loop:

- delete whitelist[toRemoveAddresses[i]] (src/DegComBase.sol#301)

Recommendation:

Use a local variable to hold the loop computation result.

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop>

Alleviation:

The Degcom Team has acknowledge the issue.

SEC-03: Functions Not Used Internally Could Be Marked External

Vulnerability Detail	Severity	Location	Category	Status
Functions Not Used Internally Could Be Marked External	Informational	Check on finding	Optimization	Resolved

Finding:

```
L35:  function getV3AmountsOut(
L271:  function withdraw() public payable onlyDev {
L279:  function close() public onlyDev {
```

Description

Several functions in the contract that are never called internally are marked as `public` instead of `external`. This is a gas optimization issue since `external` function calls are cheaper than `public` ones when the function arguments are large arrays or structs.

Recommendation:

When a function is marked as `public`, Solidity copies array arguments to memory, even if the function is called externally. However, external functions can read array arguments directly from `calldata`, saving gas.

References: Solidity Language Document: Function Visibility
<https://docs.soliditylang.org/en/latest/contracts.html#function-visibility>

Alleviation:

The Degcom Team has already resolved the issue.

SEC-04: Return Values of approve() Not Checked

Vulnerability Detail	Severity	Location	Category	Status
Return Values of approve() Not Checked	Informational	Check on finding	Naming Conventions	Resolved

Finding:

```
L167: IBEP20(path[0]).approve(dexAddress, feeParams[5]);
```

Description

The contract does not check the return value of the `approve()` function call for IBEP20/ERC20 tokens. Some token implementations return false instead of reverting on failure, which could lead to silent failures and subsequent transaction issues.

Recommendation:

ERC20 standard specifies that the `approve()` function should return a boolean indicating success or failure. Some token implementations (notably USDT) don't revert on failure but instead return false.

Use OpenZeppelin SafeERC20 to enhance the accuracy of return values checked.

Alleviation:

The Degcom Team has already resolved the issue.

GAS-01: Cache array length outside of loop

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

Finding:

```
L43:      for (uint256 i; i < pool.length; i++) {  
L76:      for (uint i = 0; i < tokenPath.length; i++) {  
L140:     for (uint i = 0; i < tokenPath.length; i++) {  
L206:     for (uint i = 0; i < tokenPath.length; i++) {  
L241:     for (uint i = 0; i < tokenPath.length; i++) {  
L289:     for (uint i = 0; i < toAddAddresses.length; i++) {  
L300:     for (uint i = 0; i < toRemoveAddresses.length; i++) {
```

Description

The contract contains multiple instances where array lengths are read within loop conditions without being cached. This results in unnecessary gas consumption due to repeated storage (SLOAD) or memory (MLOAD) operations in each iteration.

Recommendation:

When iterating over an array in Solidity, reading the array length in each iteration can lead to unnecessary gas costs. Caching the array length outside the loop can optimize gas usage. This applies to both storage arrays (which incur sload operations) and memory arrays (which incur mload operations).

Reference: Solidity Layout in memory
https://github.com/ethereum/solidity/blob/develop/docs/internals/layout_in_memory.rst

Alleviation:

The Degcom Team has acknowledge the issue.

GAS-02: Use Custom Errors

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

Finding:

```
L76:         for (uint i = 0; i < tokenPath.length; i++) {
L140:        for (uint i = 0; i < tokenPath.length; i++) {
L206:        for (uint i = 0; i < tokenPath.length; i++) {
L241:        for (uint i = 0; i < tokenPath.length; i++) {
L289:        for (uint i = 0; i < toAddAddresses.length; i++) {
L300:        for (uint i = 0; i < toRemoveAddresses.length; i++) {
```

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:

The Degcom Team has acknowledge the issue.

GAS-03: Functions guaranteed to revert when called by normal users can be marked payable

Vulnerability Detail	Severity	Location	Category	Status
Functions guaranteed to revert when called by normal users can be marked payable	Gas-optimization	Check on finding	Gas Optimization	Resolved

Finding:

```
L279:    function close() public onlyDev {
```

Recommendation:

Functions that are restricted to certain roles (e.g., onlyOwner) and will revert if called by normal users can be marked as payable. This reduces gas costs for legitimate callers by eliminating the need for the compiler to include checks for whether a payment was provided.

Mark the close functions as payable. This optimization will reduce the gas cost for the owner when these functions are called.

Alleviation:

The Degcom Team has already resolved the issue.

GAS-04: `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

Vulnerability Detail	Severity	Location	Category	Status
<code>++i</code> costs less gas than <code>i++</code> , especially when it's used in <code>for</code> -loops (<code>--i</code> / <code>i--</code> too)	Gas-optimization	Check on finding	Gas Optimization	Resolved

Finding:

```

L43:      for (uint256 i; i < pool.length; i++) {
L76:      for (uint i = 0; i < tokenPath.length; i++) {
L140:     for (uint i = 0; i < tokenPath.length; i++) {
L206:     for (uint i = 0; i < tokenPath.length; i++) {
L241:     for (uint i = 0; i < tokenPath.length; i++) {
L289:     for (uint i = 0; i < toAddAddresses.length; i++) {
L300:     for (uint i = 0; i < toRemoveAddresses.length; i++) {

```

Recommendation:

Using `++i` (pre-increment) instead of `i++` (post-increment) can save gas, especially in `for` loops. The same principle applies to decrement operations (`--i` vs `i--`).

Change post-increment `i++` to pre-increment `++i` to optimize gas usage.

Alleviation:

The Degcom Team has already resolved the issue.

GAS-05: Use != 0 instead of > 0 for unsigned integer comparison

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

Finding:

```
L68:         require(msg.value > 0, "insufficient funds");  
  
L132:        require(feeParams[5] > 0, "insufficient funds");
```

Recommendation:

Using != 0 for checking if an unsigned integer is greater than zero can save gas compared to using > 0.

Alleviation:

The Degcom Team has already resolved the 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



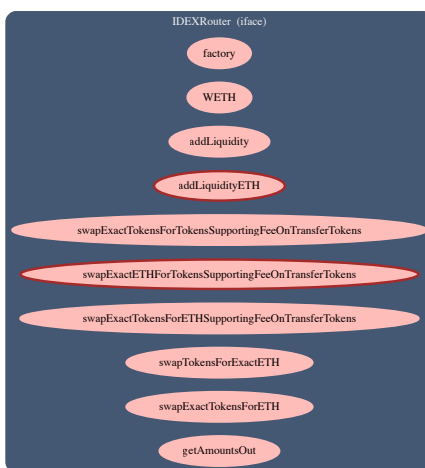
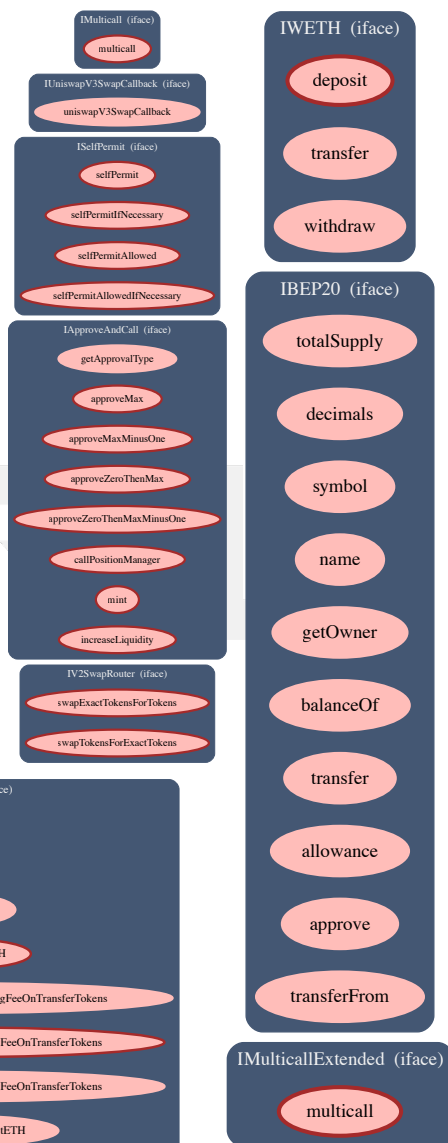
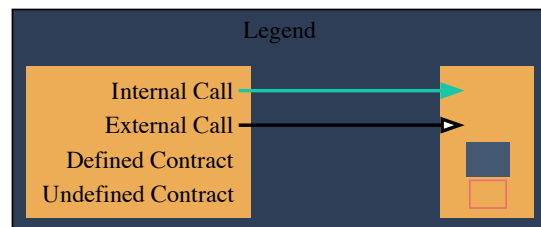
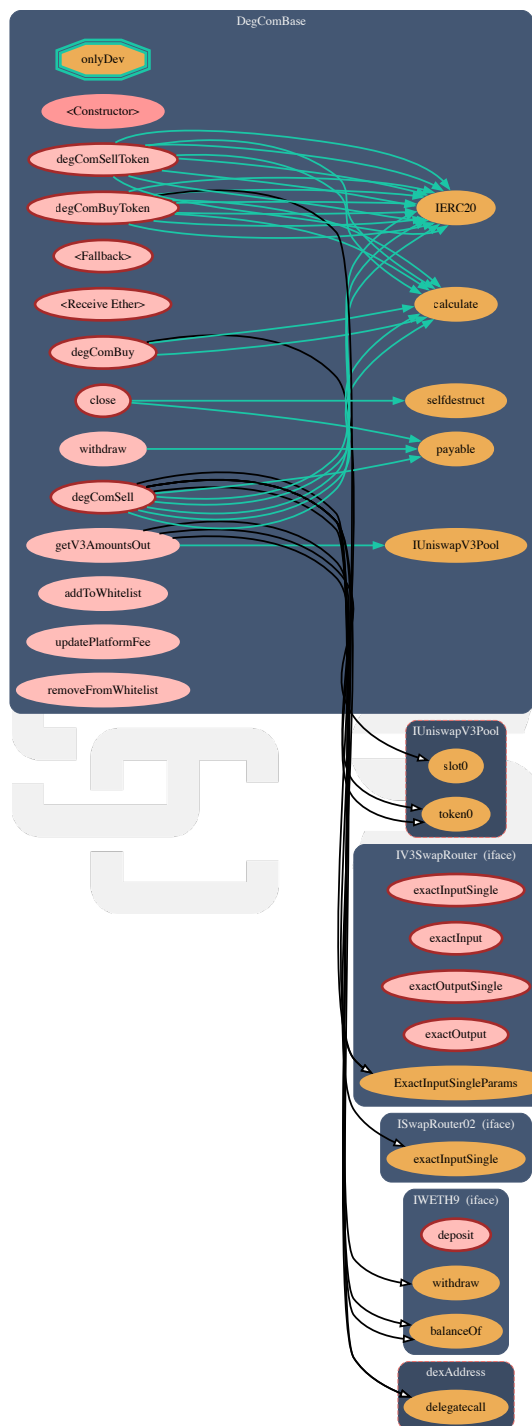
Contracts Description Table

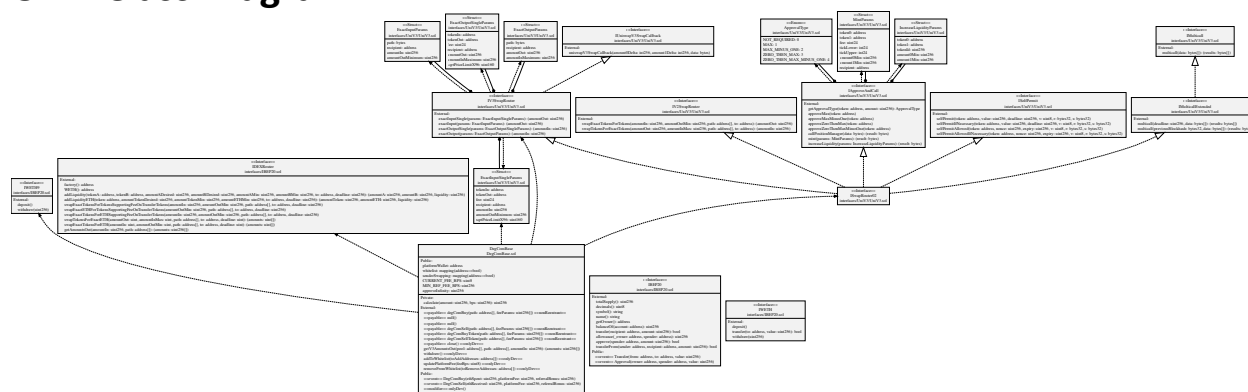
Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
DegComBase	Implementation	ReentrancyGuard		
L		Public !	🔴	NO !
L	calculate	Private 🗝️		
L	getV3AmountsOut	External !		NO !
L	degComBuy	External !	💰	nonReentrant
L		External !	💰	NO !
L		External !	💰	NO !
L	degComSell	External !	💰	nonReentrant
L	degComBuyToken	External !	💰	nonReentrant
L	degComSellToken	External !	💰	nonReentrant
L	withdraw	External !	🔴	onlyDev
L	close	External !	💰	onlyDev
L	addToWhitelist	External !	🔴	onlyDev
L	updatePlatformFee	External !	🔴	onlyDev
L	removeFromWhitelist	External !	🔴	onlyDev

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Call Graph





About SCRL

SCRL (Previously name SECURI LAB) was established in 2020, and its goal is to deliver a security solution for Web3 projects by expert security researchers. To verify the security of smart contracts, they have developed internal tools and KYC solutions for Web3 projects using industry-standard technology. SCRL was created to solve security problems for Web3 projects. They focus on technology for conciseness in security auditing. They have developed Python-based tools for their internal use called WAS and SCRL. Their goal is to drive the crypto industry in Thailand to grow with security protection technology.



Support ALL EVM L1 - L2

Smart Contract Audit

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

scrl.io



Follow Us On:

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