SAULIDITY

DESPACE
PROTOCOL

2022 | SMART CONTRACT SECURITY ANALYSIS

PREPARED BY
Saulidity

PRESENTED TO
DeSpace Protocol

2022

SAULIDITY

SECURITY REPORT

Smart Contract Audit

saulidity.com

Saulidity

@Saulidity

DISCLAIMER

This report does not constitute financial advice, and Saulidity is not accountable or liable for any negative consequences resulting from this report, nor may Saulidity be held liable in any way. You agree to the terms of this disclaimer by reading any part of the report. If you do not agree to the terms, please stop reading this report immediately and delete and destroy any and all copies of this report that you have downloaded and/or printed. This report was entirely based on information given by the audited party and facts that existed prior to the audit. Saulidity and its auditors cannot be held liable for any outcome, including modifications (if any) made to the contract(s) for the audit that was completed. No modifications have been made to the contract(s) by the Saulidity team, but if it does, it will be indicated explicitly. The audit does not include the project team, website, logic, or tokenomics, but if it does, it will be indicated explicitly. The security is evaluated only on the basis of smart contracts only. There were no security checks performed on any apps or activities. There hasn't been a review of any product codes. It is assumed by Saulidity that the information and materials given were not tampered with, censored, or misrepresented. Even if this report exists and Saulidity makes every effort to uncover any security flaws, you should not rely completely on it and should conduct your own independent research. Saulidity hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saulidity, for any amount or kind of loss or damage that may result to you or any other person or any kind of company, community, association and institution. Saulidity is the exclusive owner of this report, and it is published by Saulidity. Without Saulidity's express written authorization, use of this report for any reason other than a security interest in the individual contacts, or use of sections of this report, is forbidden.

Table of Contents

Saulidity

Saulidity is a renowned cybersecurity firm specializing in the analysis and development of Smart contracts. Saulidity, as a full-service security organization, can help with a variety of audits and project development.

In a market where confidence and trust are key, a genuine project may simply increase its user base enormously with an official audit performed by Saulidity.

Introduction

For a thorough understanding of the audit, please read the entire document.

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.
The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

During our audit, we conducted a thorough inquiry using automated analysis and manual review approaches.

The security specialists did a complete study independently of one another in order to uncover any security issues in the contracts as comprehensively as feasible. For optimum security and professionalism, all of our audits are undertaken by at least two independent auditors.

The audit was carried out on contracts that had not yet been deployed. The project's website, logic, or tokenomics have not been vetted by the Saulidity team.

Scope

We analyze smart contracts for both well-known and more specific vulnerabilities.

Here are some of the most well-known vulnerabilities that are taken into account but not limited to:

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Audit & Project Information

Project Name	DeSpace Protocol
Contract Name	DeSwapGovernanceToken.sol DeSwapFarm.sol
Report ID	SAUL52200 V1.0
Website	despace.io
Contact	Obasi Co-Founder
Contact Information	TG @francisobasi1
Code language	Solidity

Summary Table

SEVERITY	FOUND
Critical	0
High	0
Medium	0
Low	3
Lowest / Code Style / Optimized Practice	1

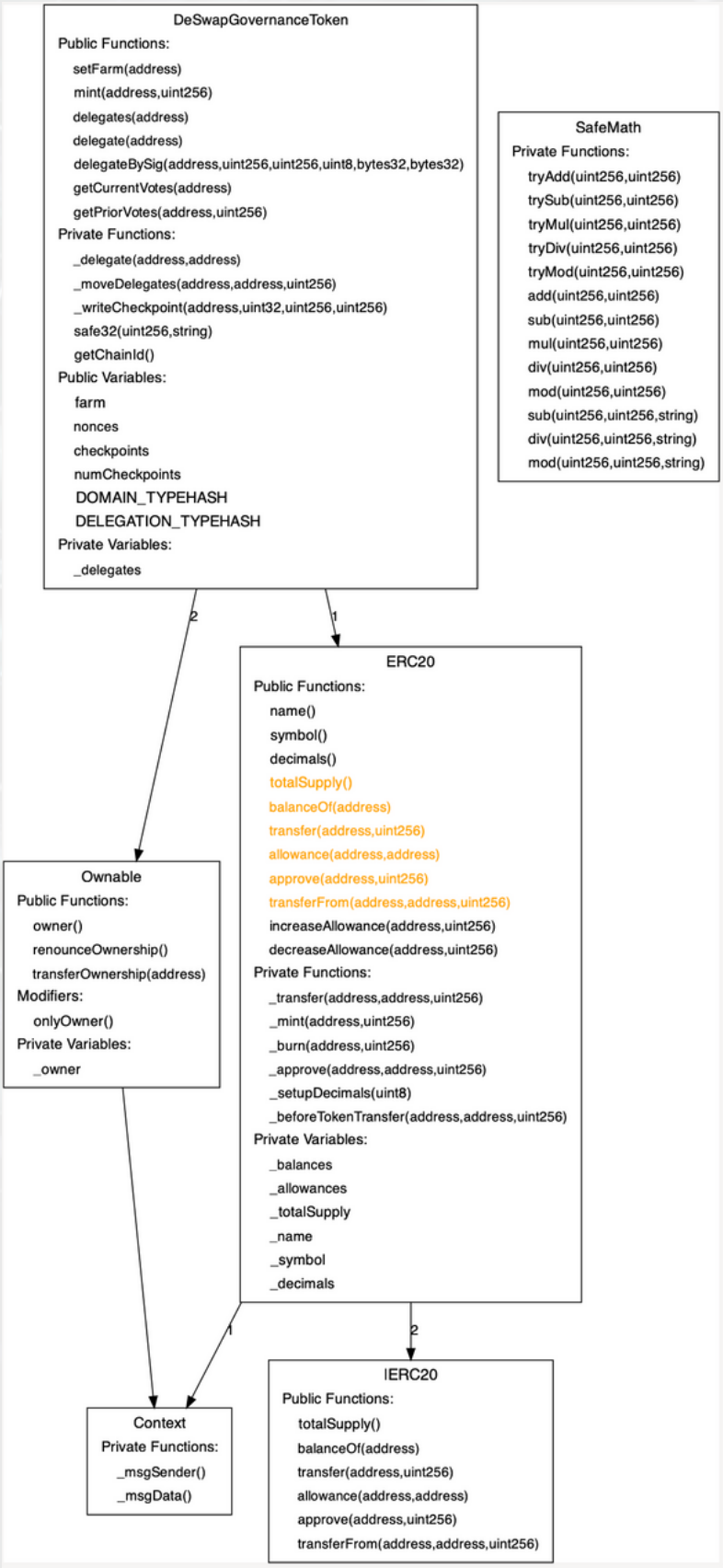
Executive Summary

ACCORDING TO THE ANALYSIS, **THERE ARE NO CRITICAL, HIGH, OR MEDIUM SEVERITY SECURITY VULNERABILITIES.**

ALL ISSUES FOUND DURING AUTOMATED ANALYSIS WERE MANUALLY REVIEWED, AND FALSE POSITIVES WERE ELIMINATED. THE FINDINGS ARE PRESENTED IN THE ANALYSIS SECTION OF THE REPORT.

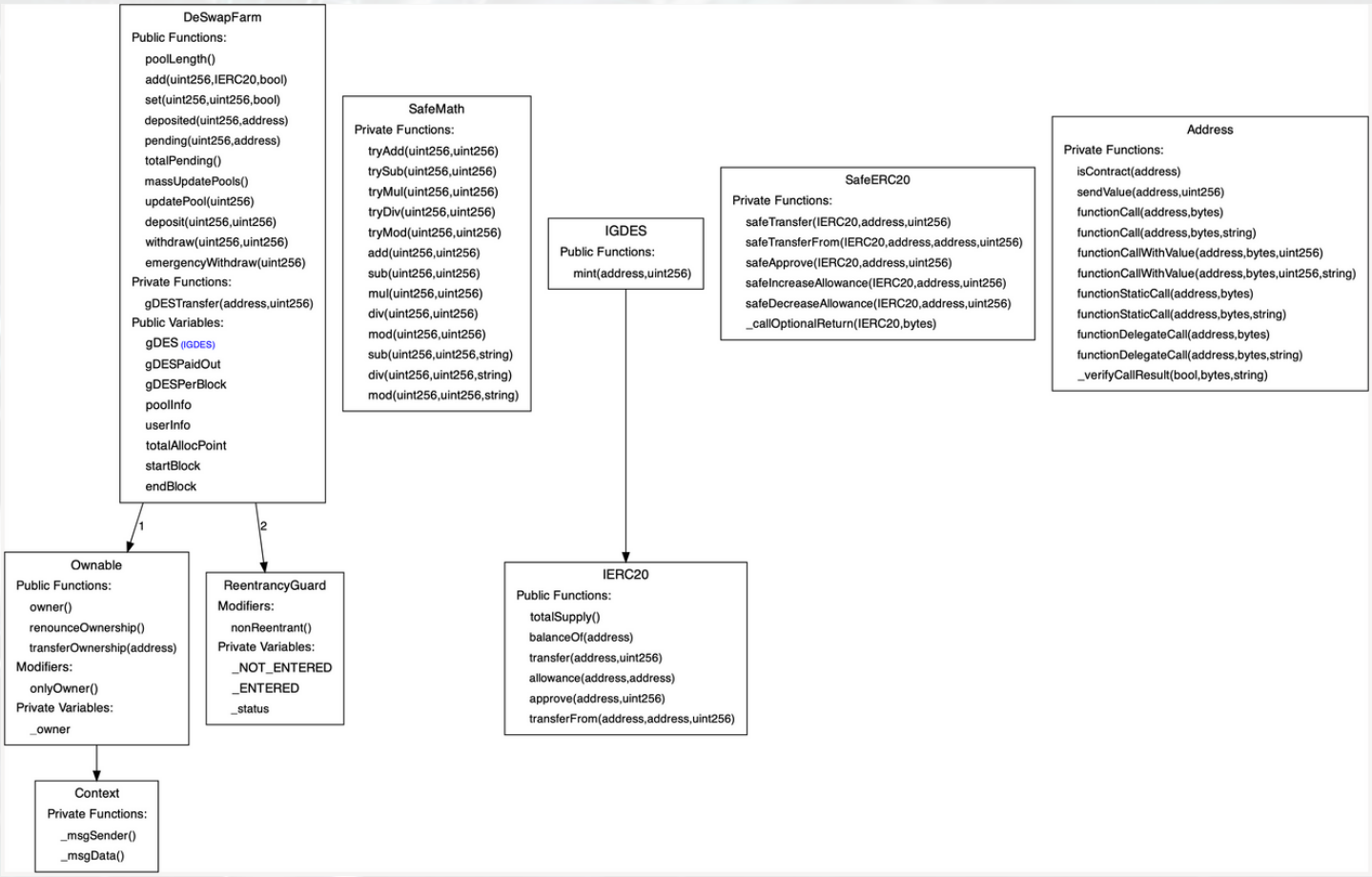
Inheritance

DeSwapGovernanceToken.sol



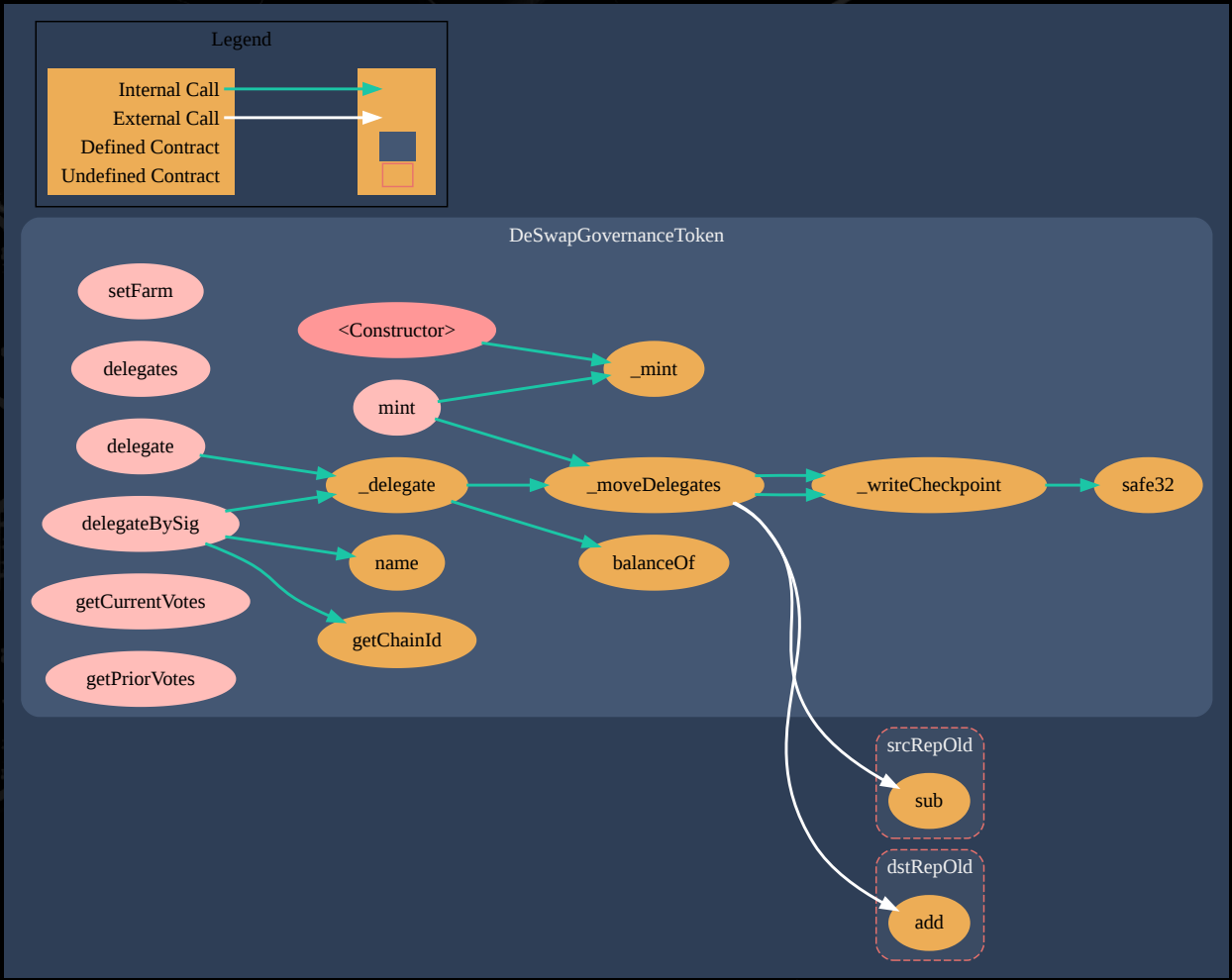
Inheritance

DeSwapFarm.sol



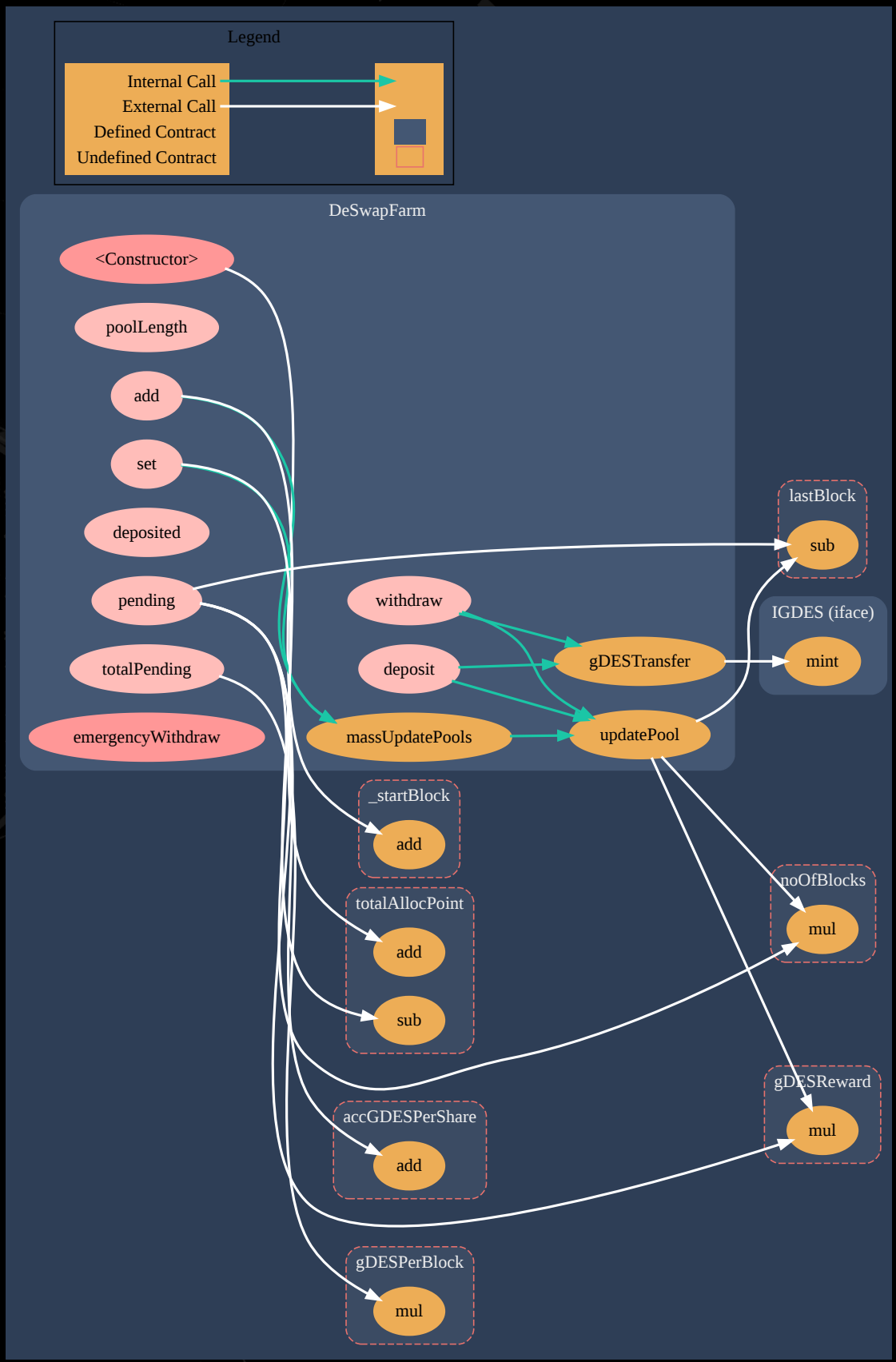
Call Graph

DeSwapTokenGovernance.sol



Call Graph

DeSwapTokenFarm.sol



Analysis

DeSwapGovernanceToken.sol

Issue: block.timestamp

Severity: Low

Location: L111-150

Description: Relying on block.timestamp for randomness which can be manipulated by miners.

```
DeSwapGovernanceToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)
 uses timestamp for comparisons
 Comparisons:
 - require(bool,string)(now <= expiry,DeSwapGovernanceToken::delegateBySig: signature
expired)
```

Comment: We recommend to avoid relying on block.timestamp. A good rule of thumb in evaluating timestamp usage is that if the scale of your time-dependent event can vary by 15 seconds and maintain integrity, it is safe to use.

Analysis

DeSwapFarm.sol

Issue: Missing events arithmetic

Severity: Low

Location: L91-111, L114-126

Description: Events for critical arithmetic parameters are missing.

```
DeSwapFarm.add(uint256,IERC20,bool) should emit an event for:
 - totalAllocPoint = totalAllocPoint.add(_allocPoint)
DeSwapFarm.set(uint256,uint256,bool) should emit an event for:
 - totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint)
```

Comment: We recommend emitting an event for critical parameter changes.

Analysis

DeSwapFarm.sol

Issue: Division before multiplication

Severity: Low

Location: L139-166, L187-210

Description: Integer division may result in a truncation. As a result, executing a multiplication before division can help to minimize accuracy loss in some cases.

```
DeSwapFarm.pending(uint256,address) performs a multiplication on the result of a division:
 -gDESReward = noOfBlocks.mul(gDESPerBlock).mul(pool.allocPoint).div(totalAllocPoint)
 -accGDESPerShare = accGDESPerShare.add(gDESReward.mul(1e36).div(lpSupply))
DeSwapFarm.updatePool(uint256) performs a multiplication on the result of a division:
 -gDESReward = noOfBlocks.mul(gDESPerBlock).mul(pool.allocPoint).div(totalAllocPoint)
 -pool.accGDESPerShare = pool.accGDESPerShare.add(gDESReward.mul(1e36).div(lpSupply))
```

Comment: We recommend to consider doing multiplication first, and then division.

Analysis

DeSwapFarm.sol

Issue: Possibility to declare public function as external.

Severity: **Lowest / Code Style / Optimized Practice**

Location: L257-264

Description: In order to save gas, public functions that are never called by the contract could be declared external.

```
emergencyWithdraw(uint256) should be declared external:
  - DeSwapFarm.emergencyWithdraw(uint256)
```

Comment: We recommend using the external attribute for functions that are never called from the contract.

Testing Standards

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.
The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

The blockchain platform is used to deploy and execute smart contracts. The platform, its programming language, and other smart contract-related applications all have vulnerabilities that may be exploited. As a result, the audit cannot ensure the audited smart contract(s) explicit security.Audits can't make statements or warranties on security of the code.It also cannot be deemed an adequate assessment of the code's utility and safety, bug-free status, or any statements of the smart contract.
While we did our best in completing the study and publishing this report, it is crucial to emphasize that you should not rely only on it; we advocate all projects doing many independent audits and participating in a public bug bounty program to assure smart contract security.

Testing Standards

1. Gather all relevant data.

2. Perform a preliminary visual examination of all documents and contracts.

3. Find security holes with specialist tools & manual review with independent experts.

4. Create and distribute a report.

SAULIDITY

Smart Contract
Audit

saulidity.com
Saulidity
@Saulidity