Swapp.ee

Pool (Farm & Staking)

SMART CONTRACT AUDIT

19.07.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Decentralized LLC (Swapp.ee). If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (15.07.2021)	Layout
0.2 (15.07.2021)	Test Deployment
0.5 (16.07.2021)	Automated Security Testing Manual Security Testing
0.6 (16.07.2021)	Testing SWC Checks
0.7 (16.07.2021)	Verify Claims
0.9 (16.07.2021)	Summary and Recommendation
1.0 (16.07.2021)	Final document
1.1 (19.07.2021)	Added deployed contract addresses

2. About the Project and Company

Company address:

Decentralized LLC
420 N Wabash Ave, Ste 520
Chicago IL, 60611
USA

Website: https://swapp.ee

Twitter: https://twitter.com/SwappFi

Telegram: https://t.me/SwappToken

GitHub: https://github.com/Swapp-Token

2.1 Project Overview

SWAPP Protocol is an Ethereum blockchain ERC-20 smart contract. SWAPP is a decentralized, fairly launched, ETH-paired utility token used to both facilitate yield farming rewards in the Swapp DeFi platform as well as serve as the form of rewards within the Swapp smartphone app (on ios and android).

Swapp is democratizing the data industry, putting the power over consumer data where it belongs: In the hands of each individual consumer. Powering this newly decentralized data industry is our blockchain-driven infrastructure which is open to users ranging in size from individual consumers up to institutional players. As millions of consumers begin capitalizing on their own data monetization, this puts upward pressure on the price of each compensation token rewarded. Token holders will also benefit from the optional liquidity pools structured to pay reward to those who choose to “stake” their SWAPP tokens, with dividends as high as 50% APY for early adopters.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

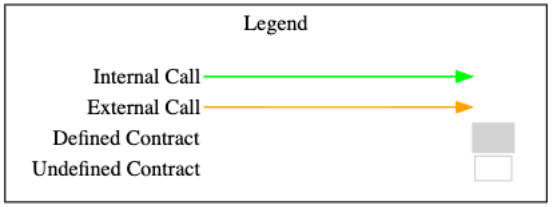
Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.2.0/contracts/access/Ownable.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.2.0/contracts/security/ReentrancyGuard.sol
@openzeppelin/contracts/utils/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.2.0/contracts/utils/math/SafeMath.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.2.0/contracts/token/ERC20/IERC20.sol

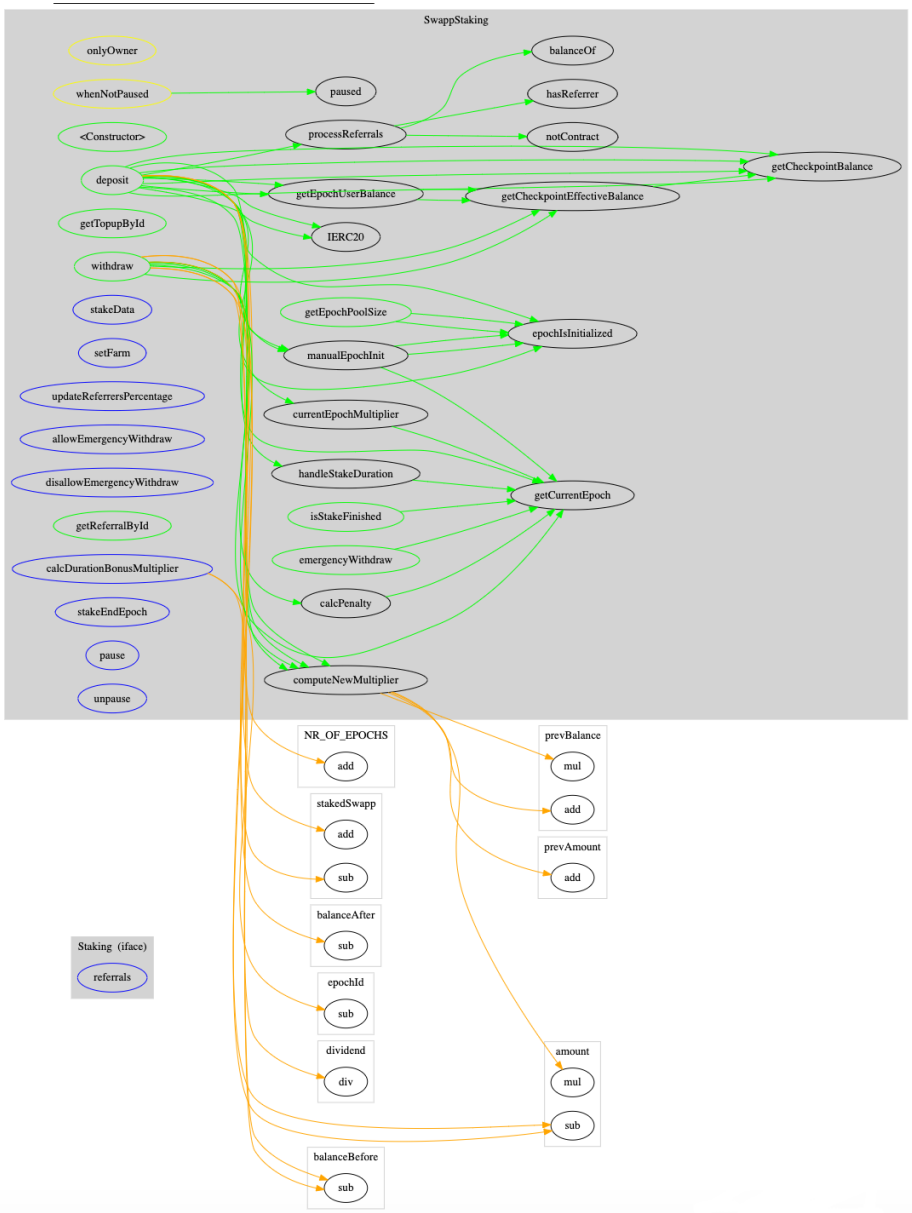
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

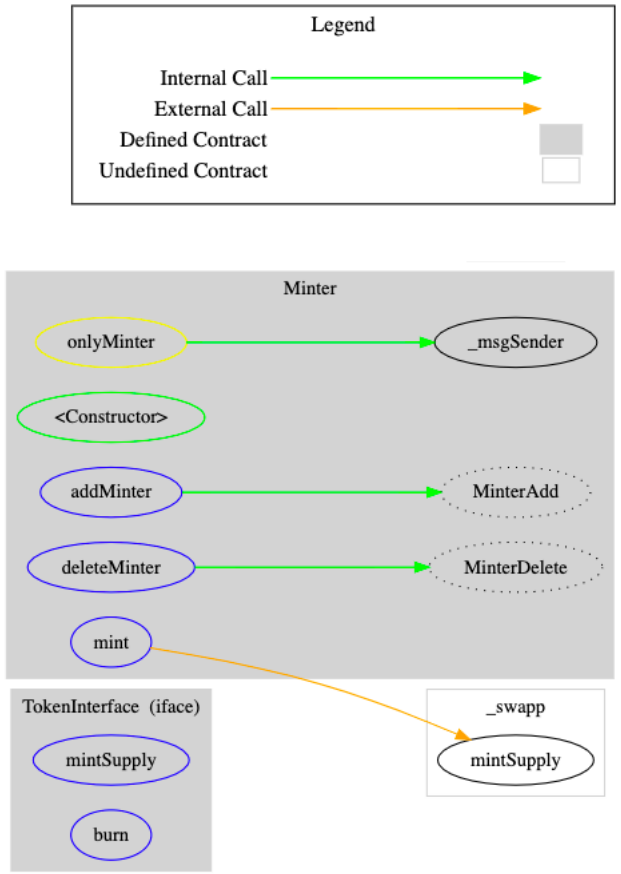
File	Fingerprint (MD5)
SwappStaking.sol	6095fe40cb3a3af7afb0b02b9e028da9
SwappYieldFarm.sol	e799207a812c9fb3be2908bbb3a83e71
Minter.sol	07e2a2d66ee677df3a7ee498ace3fe34

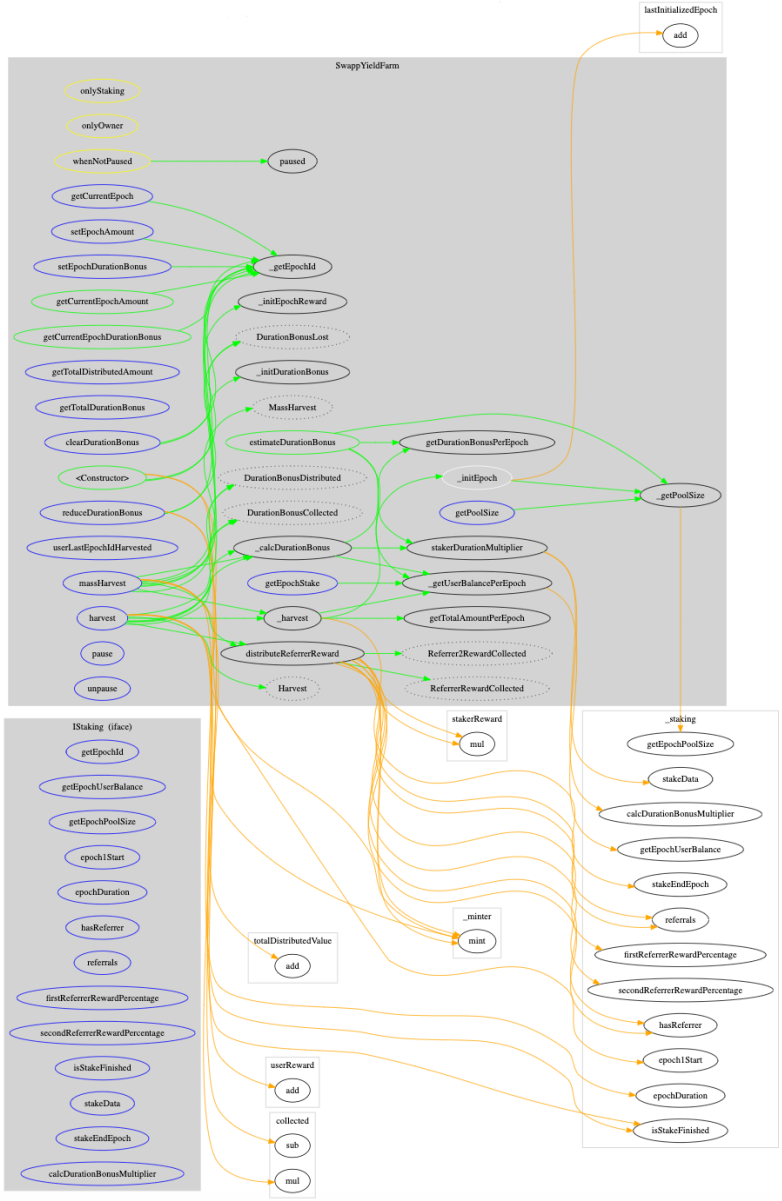
4.4 Metrics / CallGraph (SwappStaking)



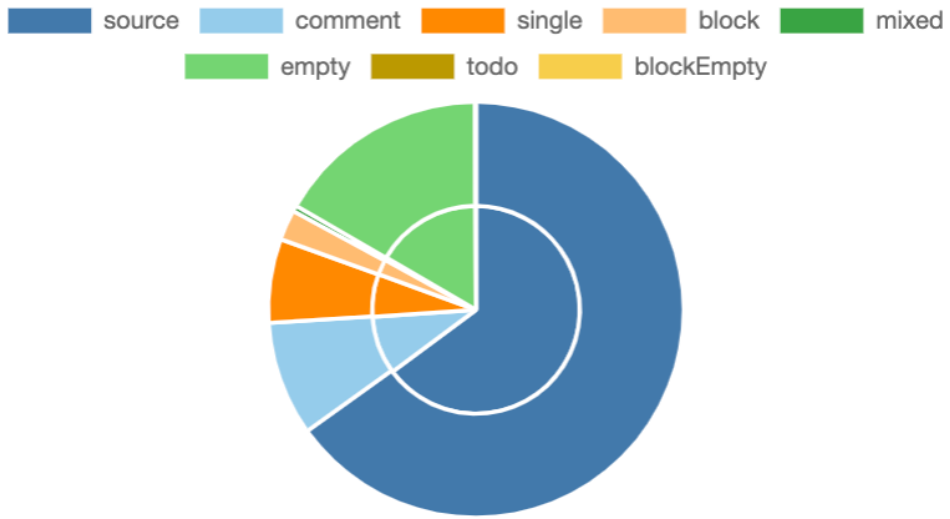


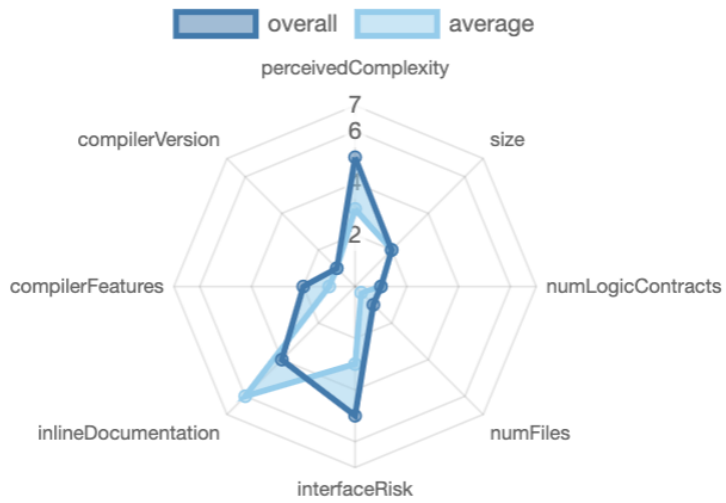
4.4.1 Metrics / CallGraph (SwappYieldFarm & Minter)





4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`0.8.0`			`yes` (1 asm blocks)	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2

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
71	0

External	Internal	Private	Pure	View
45	53	0	3	46

StateVariables

Total	Public
40	22

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝🔍	contracts/Swapp/Minter.sol	1	1	41	38	27	1	23	
📝🔍	contracts/Staking/SwappYieldFarm.sol	1	2	494	476	375	36	258	
📝🔍	contracts/Staking/SwappStaking.sol	1	4	650	636	426	80	315	🖥
📝🔍	**Totals**	**3**	**7**	**1185**	**1150**	**828**	**117**	**596**	🖥

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, ...)

5. Scope of Work

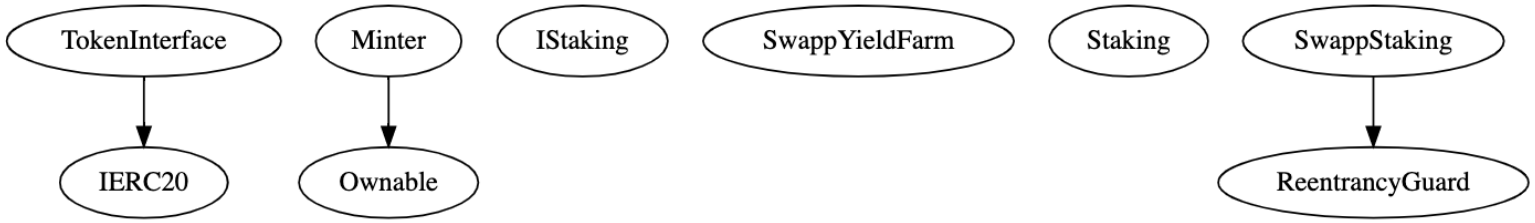
The Swapp.ee Team provided us with the files that needs to be tested. The scope of the audit are the Farm & Staking contracts.

Following contracts with the direct imports has been tested:
- SwappYieldFarm.sol
- SwappStaking.sol
- Minter.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES
During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES
During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES
During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract.

INFORMATIONAL ISSUES

5.1.1 No dependency from OpenZeppelin via package.json
Severity: INFORMATIONAL
Status: FIXED
File(s) affected: package.json

Attack / Description	Code Snippet	Result/Recommendation

OpenZeppelin libraries have been imported via `import "@openzeppelin/contracts/token/ERC20/IERC20.sol";`	NA	Consider to install the correct OpenZeppelin dependency, via package.json
		
		`"dependencies": {`
		`…`
		`"@openzeppelin/contracts": "4.2.0"`

within the contract, but no npm dependency via package.json is defined.		... }

5.2. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	X
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✅
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debrief took place on the July 16, 2021. The overall code quality of the project is very good and the reentrancy guard was implemented correctly, which decrease the attack surface. It correctly implemented widely-used and reviewed contracts from OpenZeppelin.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, no critical issues were found, after the manual and automated security testing and the claim have been successfully verified.

7. Deployed Smart Contract

VERIFIED

SwappStaking
https://etherscan.io/address/0x60F4D3e409Ad2Bb6BF5edFBCC85691eE1977cf35#code

SwappYieldFarm
https://etherscan.io/address/0x51fac321561C8AE715F0A3113fFCb4E592203Da4#code

Minter
https://etherscan.io/address/0xBC1f9993ea5eE2C77909bf43d7a960bB8dA8C9B9#code