



Scifi.Finance (SCIFI INDEX & SPICE Token)

SMART CONTRACT AUDIT

20.12.2020

Made in Germany by Chainsulting.de



1. Disclaimer.....	3
2. About the Project and Company	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology	7
4.2 Used Code from other Frameworks/Smart Contracts (direct imports).....	8
4.3 Tested Contract Files (SPICE Token).....	9
5. Scope of Work.....	15
5.1 Manual and Automated Vulnerability Test.....	16
5.1.1 SPDX license identifier missing	16
5.1.2 A floating pragma is set.....	17
5.1.3 Constant name must be in capitalized letters (Style Guide)	18
5.3. SWC Attacks	21
5.4 Verifying claims	25
5.4.1 Regards the SCIFI POC no smart contracts are directly deployed, everything is deployed via customs scripts over the setprotocol's creator contract [PASSED]	25
5.4.2 Checking the usage of the right smart contracts [PASSED]	25
5.4.3 Checking that we're interacting with the correct contracts [PASSED].....	25
5.4.4 Checking the security of the custom scripts [PASSED].....	27
5.4.5 Checking the overall security of the SPICE governance token [PASSED].....	27
6. Executive Summary.....	28
7. Deployed Smart Contract	29

1. Disclaimer

The audit makes no statements or warranties 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 Scifi.Finance. 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 (02.12.2020)	Layout
0.5 (03.12.2020)	Automated Security Testing Manual Security Testing
0.6 (04.12.2020)	
0.8 (05.12.2020)	Testing SWC Checks
1.0 (06.12.2020)	Summary and Recommendation
2.0 (07.12.2020)	Final document
2.1 (16.12.2020)	Adding SPICE Token
2.2 (19.12.2020)	Adding SCIFI Token
2.3 (20.12.2020)	Checking all claims (PASSED)

2. About the Project and Company

Company address: Anon

Website: <https://scifi.finance/>

GitHub: <https://github.com/spicedao>

Twitter: https://twitter.com/sci_finance

Discord: <https://discord.gg/5dPXRWuD4n>

Telegram: https://twitter.com/sci_finance

Medium: <https://scifinance.medium.com/>

2.1 Project Overview

The SCIFI mission assembles veteran engineers, seasoned experts and the most daring citizens of crypto space to discover and grow the most valuable gems in the vast reaches of the universe and to harvest the spoils of the adventure in the form of a governance token, SPICE. SCIFI represents a hand-curated basket of gems, automatically rebalanced on a monthly basis. SCIFI is 100% collateralised and tradable on any ERC20-enabled DEX. SCIFI and any potential future missions are governed by the community. Supporting the SCIFI mission by holding SCIFI, providing liquidity, promoting it or providing essential infrastructure is rewarded with a governance token called SPICE. SCIFI combines the best of passive investment strategies, expert curation and decentralised ledger technology:

- DAO governance: SCIFI's lift-off is guided by Mission Control and the Galactic Council, but will be community-governed after the first 6 months.
- Experts: The 'Galactic Council' is the representative body that joins Mission Control in the creation of SCIFI.
- Diversified investing: A single token provides exposure to a basket of vetted tokens, automatically rebalanced.
- Zero fees: No fees will be charged. SCIFI is free and always will be.
- Secure: SCIFI is 100% collateralised, audited and open-source.



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)

SCIFI INDEX

SCIFI is the index token based off Set v2

<https://github.com/SetProtocol/set-v2>

SPICE Token

SPICE is the governance token based off INDEX from Indexcoop

<https://github.com/SetProtocol/index-coop/>

<https://github.com/Uniswap/merkle-distributor/blob/master/contracts/MerkleDistributor.sol>

<https://github.com/Uniswap/merkle-distributor/blob/master/contracts/interfaces/IMerkleDistributor.sol>

<https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol>

<https://github.com/Synthetixio/synthetix/blob/develop/contracts/RewardsDistributionRecipient.sol>

Including the following imports

1. SafeMath.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol>

2. Math.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/Math.sol>

3. SafeERC20.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/SafeERC20.sol>

4. ERC20.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol>

5. IERC20.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol>

6. MerkleProof.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/cryptography/MerkleProof.sol>

7. ReentrancyGuard.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol>

4.3 Tested Contract Files (SPICE Token)

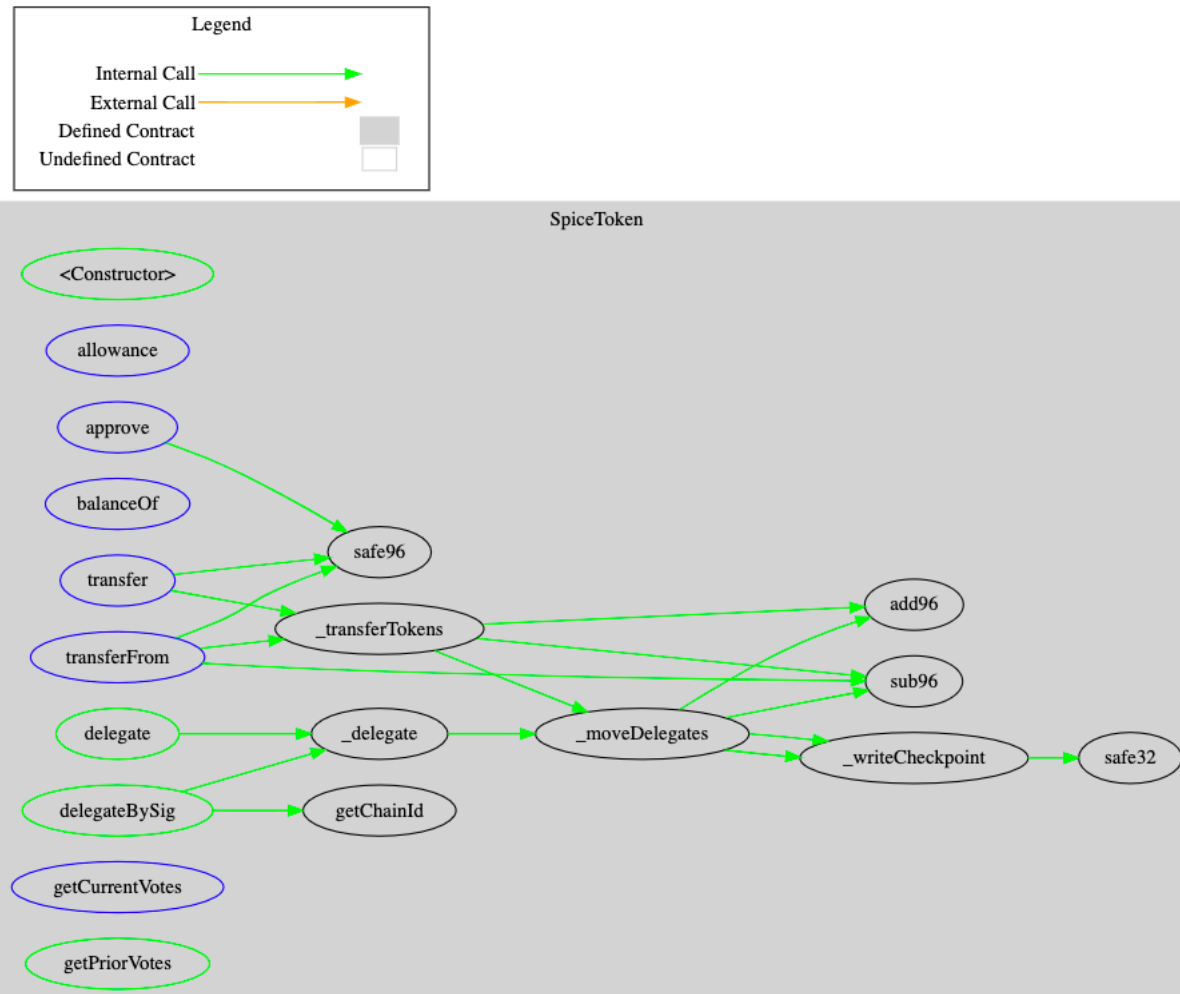
The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

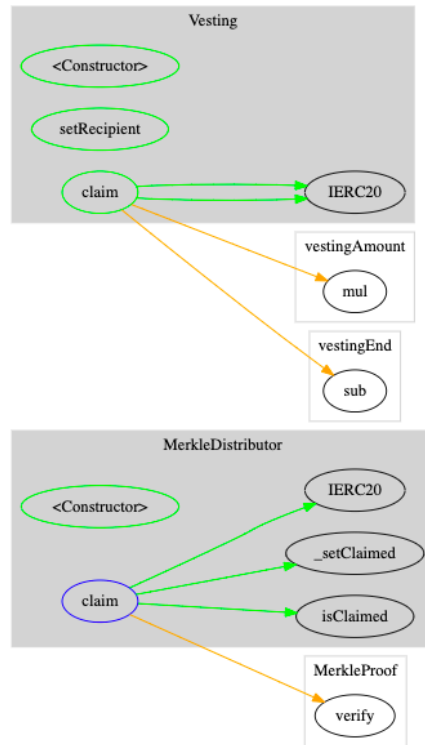
File	Fingerprint (SHA256)
token/MerkleDistributor.sol	08a77ebef3ffc8f219138ab7e7f64962cfb217d41bf2b15f67ec496e26c66ca1
token/SpiceToken.sol	021831d9692d2b38395ea77670526f8998c7707657d5968769f6da97e772cb13
token/Vesting.sol	9d3f2d5798347259beea7801409663963cbc151e4c7a40a33d1299f3912259c3
staking/StakingRewards.sol	069b60bbe3659c49f236673eea0c020841cd171a35f8f7a20f5b37260a559589
staking/RewardsDistributionRecipient.sol	ff90697c320f20db1313061d28f0d702210814ee506e5f1c80ede1e81bd1c3c6
interfaces/IMerkleDistributor.sol	3867639b55d679084905e9d87ef7fc168311bef2632cecc00b009e82ecd5fd3f

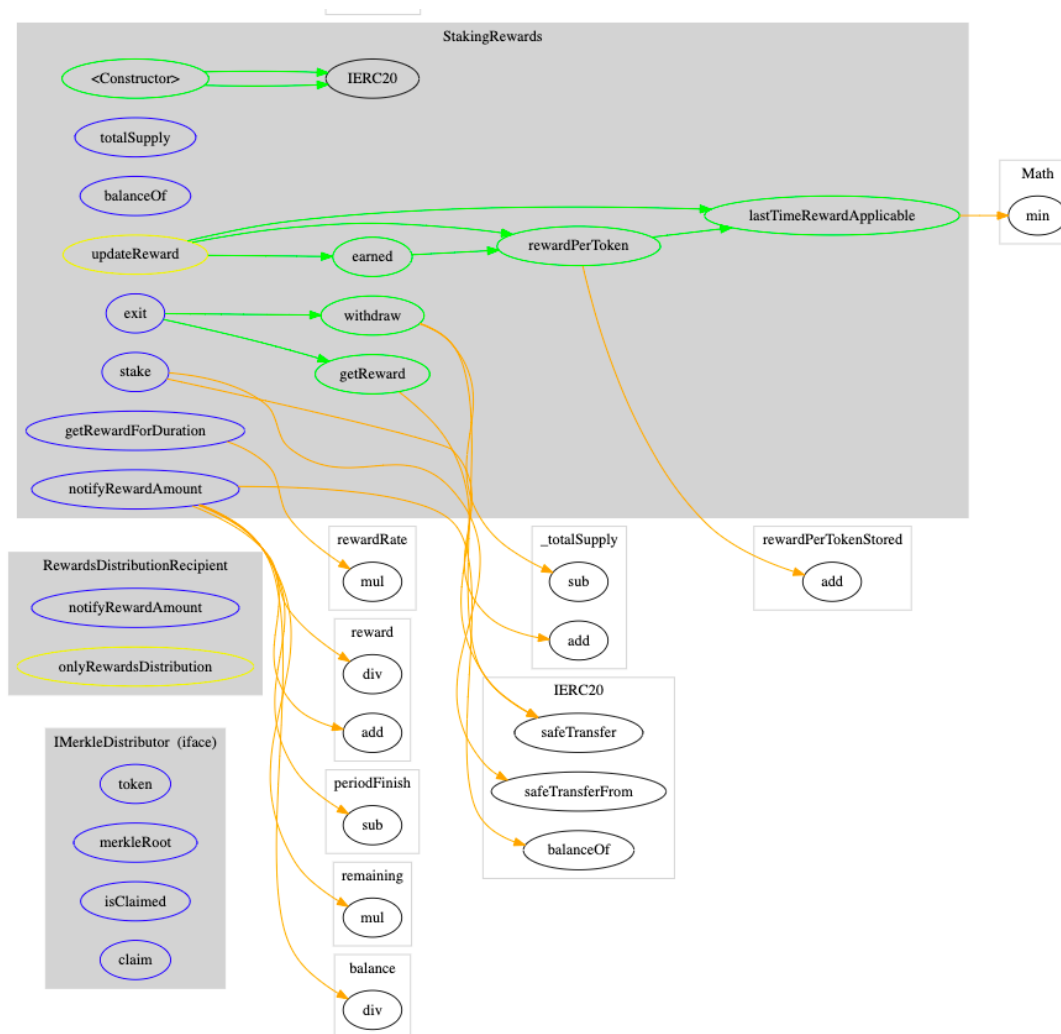
4.4 Tested Scripts (SCIFI INDEX)

File	Fingerprint (SHA256)
scripts/poc.js	b57680218c836495f2667401e22705d6ab9c231733c0afa7e7297204295e3c1c
src/highLevelFunctions.js	384543b17e7e89d1250cb1f438a48d38f7dc03846f540cb29b059e5acdac38f2
src/lowLevelFunctions.js	e3aa060804cc4d563e60914fb5ed4e2812ae9137e468cb8521365d36e9ba30ea
configs/constants.json	1914248221e41cda9f2e4eb01008146547155f114c87f1bdbae06fd1e3ff070e
configs/protocol-constants.json	c3ab300c13c9945a8c0a2506de687b2013fd6b9cef0757814600bc210de05f50
abis/BasicIssuanceModule.json	255180d32535b87cb9b58cf49aa1fe871eca5d0c0fbe79306b68b4b344dba59f
abis/SetToken.json	a980bd8ff966f896a30284655805c46f73f2eee340fc4a2dc5bba76cbe8d116c
abis/SetTokenCreator.json	944e6270ba28e2f8fa3309511613d6234eae83cfffcc1bada4860aae7d381ba16
abis/TradeModule.json	3b0c862afb0cd72985c3bbb1bd8160ad857a6c65b5da92d15002687e4d3018ae
abis/weth.json	ef60c7b37939199aa2437ec2c832688e1eb77d2b4c472fd9ab06ccf8bad71522

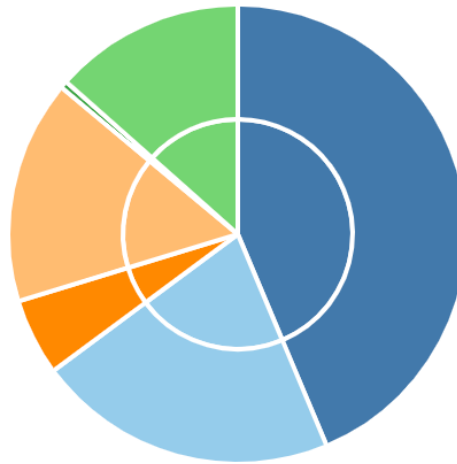
4.4 Metrics / CallGraph (SPICE Token)





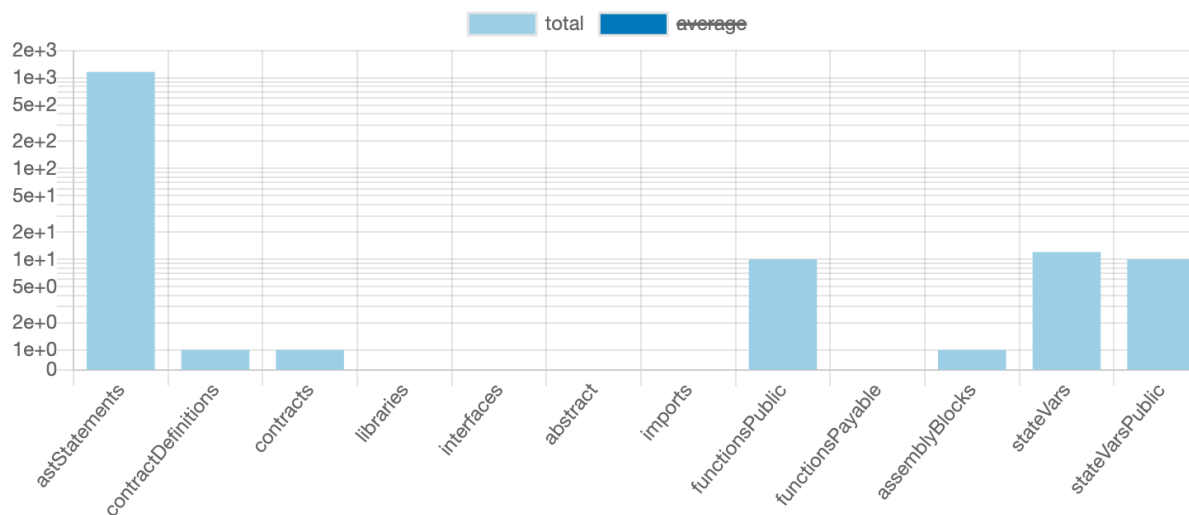


4.5 Metrics / Source Lines (SPICE Token)



4.6 Metrics / Capabilities (SPICE Token)

Solidity Versions observed	🔧 Experimental Features	💰 Can Receive Funds	💻 Uses Assembly	💣 Has Destroyable Contracts	
<code>^0.6.10</code>	<code>ABIEncoderV2</code>	<code>no</code>	<code>yes</code> (1 asm blocks)	<code>no</code>	
📡 Transfers ETH	⚡ Low-Level Calls	👤 DelegateCall	🎲 Uses Hash Functions	🔥 ECTRecover	🌀 New/Create/Create2
<code>no</code>	<code>no</code>	<code>no</code>	<code>yes</code>	<code>yes</code>	<code>no</code>



5. Scope of Work

The scifi.finance team provided us with the files that needs to be tested. The scope of the audit is the SCIFI index token based off Set v2 contracts and the SPICE governance token based off INDEX from Indexcoop.

Located at

<https://github.com/ScifiToken/scifi-contracts>

<https://github.com/ScifiToken/scifi-spice-token>

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

- Regards the SCIFI POC no smart contracts are directly deployed, everything is deployed via customs scripts over the setprotocol's creator contract
- Checking the usage of the right smart contracts
- Checking that we're interacting with the correct contracts
- Checking the security of the custom scripts
- Checking the overall security of the SPICE governance token

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

5.1.1 SPDX license identifier missing

Severity: LOW

Code: NA

Project: SPICE Token

File(s) affected: RewardsDistributionRecipient.sol, SpiceToken.sol, Vesting.sol, StakingRewards.sol

Attack / Description	Code Snippet	Result/Recommendation
SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.	NA	<pre>// SPDX-License-Identifier: UNLICENSED</pre> https://docs.soliditylang.org/en/v0.6.8/layout-of-source-files.html

INFORMATIONAL ISSUES

5.1.2 A floating pragma is set

Severity: INFORMATIONAL

Code: SWC-103

File(s) affected: all

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is <code>^0.6.10</code> ; It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.	Line: 1 <code>pragma solidity ^0.6.10;</code>	It is recommended to follow the example (0.6.10), as future compiler versions may handle certain language constructions in a way the developer did not foresee. Not effecting the overall contract functionality.

5.1.3 Constant name must be in capitalized letters (Style Guide)

Severity: INFORMATIONAL

Code: NA

File(s) affected: SpiceToken.sol

Attack / Description	Code Snippet	Result/Recommendation
Lint: Constant name must be in capitalized SNAKE_CASE [const-name-snakecase]	Line: 6 - 10 string public constant name = "Spice";	Constants should be named with all capital letters with underscores separating words. Examples: MAX_BLOCKS, TOKEN_NAME, TOKEN_TICKER, CONTRACT_VERSION. https://docs.soliditylang.org/en/develop/style-guide.html#constants

5.2. Known attacks

Audit	Status	Audit Description
Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
Replay attack detection	Pass	After testing, there is no such safety vulnerability.
Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.
Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
Access control defect detection	Pass	After testing, there is no such safety vulnerability.
tx.progin authentication	Pass	After testing, there is no such safety vulnerability.
call injection attack	Pass	After testing, there is no such safety vulnerability.





Unverified return value call	Pass	After testing, there is no such safety vulnerability.
Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.

5.3. 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	✓
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	

5.4 Verifying claims

5.4.1 Regards the SCIFI POC no smart contracts are directly deployed, everything is deployed via customs scripts over the setprotocol's creator contract [PASSED]

We will make sure and check if the Scifi.finance team used the correct creator contract, for deployment.

5.4.2 Checking the usage of the right smart contracts [PASSED]

5.4.3 Checking that we're interacting with the correct contracts [PASSED]

Source: <https://github.com/spicedao/scifi-contracts/blob/master/configs/protocol-contracts.json>

Verify: <https://docs.tokensets.com/protocol/untitled>

`{}` protocol-contracts.json ×

configs > `{}` protocol-contracts.json > ...

```
1  {
2    "kovan": {
3      "setTokenCreatorAddress": "0xB24F7367ee8efcB5EAbE4491B42fA222EC68d411",
4      "tradeModuleAddress": "0xC93c8CDE0eDf4963ea1eea156099B285A945210a",
5      "basicIssuanceModuleAddress": "0x8a070235a4B9b477655Bf4Eb65a1dB81051B3cC1",
6      "assetLimitHookAddress": "0x0000000000000000000000000000000000000000"
7    },
8    "mainnet": {
9      "setTokenCreatorAddress": "0x65d103A810099193c892a23d6b320cF3B9E30D46",
10     "tradeModuleAddress": "0x90F765F63E7DC5aE97d6c576BF693FB6AF41C129",
11     "basicIssuanceModuleAddress": "0xd8EF3cACe8b4907117a45B0b125c68560532F94D",
12     "assetLimitHookAddress": "0x0000000000000000000000000000000000000000"
13   }
14 }
```

<https://etherscan.io/address/0x65d103A810099193c892a23d6b320cF3B9E30D46#writeContract> [PASSED]

<https://etherscan.io/address/0x90F765F63E7DC5aE97d6c576BF693FB6AF41C129#writeContract> [PASSED]

<https://etherscan.io/address/0xd8EF3cACe8b4907117a45B0b125c68560532F94D#writeContract> [PASSED]

however, they can be modified while deployment, we will recheck the contracts after deployment and give our second approval, that they are interacting with the correct contracts.

5.4.4 Checking the security of the custom scripts [PASSED]

See: 5.1 – 5.3

5.4.5 Checking the overall security of the SPICE governance token [PASSED]

See: 5.1 – 5.3

6. Executive Summary

The overall code quality of the project is very good, not overloaded with unnecessary functions. They also used widely audited contracts (4.2) which greatly benefiting the security of the overall project.

The main goal of the audit was to verify the claims regarding the security and usage of the smart contracts and deployment scripts (see the scope of work section). All claims been verified by us and passed all tests.

The utilized SetProtocol has been audited several times

https://www.setprotocol.com/pdf/chain_security_set_protocol_report.pdf

https://www.setprotocol.com/pdf/peckshield_audit_report.pdf

<https://blog.openzeppelin.com/set-protocol-audit/>

7. Deployed Smart Contract

SPICE Token

<https://etherscan.io/address/0x1fdab294eda5112b7d066ed8f2e4e562d5bcc664#code> (Verified)

SCIFI Token

<https://etherscan.io/address/0xfdc4a3fc36df16a78edcaf1b837d3acaaedb2cb4#code> (Verified)