SAFO AUDIT

SMART CONTRACT CODE REVIEW SECURITY ANALYSIS REPORT FOR LUMI FOUNDATION

TABLE OF CONTENT

OVERVIEW

$LFD

This audit is for the below maked smart contact and language of the said contract.

Project Audited	Lumi Foundation
Smart Contract Address	0x43dA25595b4b8BbceB2279d9f77EBcAa337044E8
Programming Language	Solidity
Blockchain	BSC
Type	BEP-20

SAFO Audit was requested by $LFD Lumi Foundation "SAFO Smart Contract Owner(s) to perform an Audit on there staking smart contract.

-To ensure that the smart contract serves its intended purpose, the audit's main goal was to do the following.

To identify potential security problems with the smart contract.

Utilize the reporting information in this report to assess the smart contract's risk exposure and, by resolving the issues found, enhance its security.

$LFD PROPERTIES

Smart Contract Properties

Contract Name	Lumi Foundation $LFD
Team Wallet	0x87cd02775eba9233d1b27ae5340ece05d7fbd841
Staked Token	0xe13e2b3e521080e539260d1087c087582d1bc501
Reward Token	0xe13e2b3e521080e539260d1087c087582d1bc501
WETH	0xbb4cdb9cbd36b01bd1cbaebf2de08d9173bc095c
Update At	1681325317
Total Staked	178289055.26480445629794 8064
Reward Rate	2.60524562757201646
Reward Per Token Stored	0.07092439894405919 4
Pool Start Time	1676878253
Pool End Time	1684654253

$LFD PROPERTIES

Smart Contract Properties

Pool Duration	7776000 (90 days)
Lock Period	259200 (3 days)
Early Unstake Fee	10%
Owner Address	0x1cc7eb8450cf70a8c03b09f9601d14ecdc06a0cf
Deployer Address	0x1cc7Eb8450cf70A8c03B09F9601d14ecdc06a0CF
Smart Contract	0x43dA25595b4b8BbceB2279d9f77EBcAa337044E8
Blockchain	Binance Smart Chain ~ Mainnet

CONTRACT FUNCTION

Executables

1. function claimRewards() public updateReward(msg.sender)
2. function emergencyUnstake() external updateReward(msg.sender)
3. function recoverBeans() external onlyOwner
4. function recoverWrongToken(IERC20 _token) external onlyOwner
5. function setPoolDuration(uint _duration) external onlyOwner
6. function setPoolRewards(uint _amount) external onlyOwner updateReward(address(0))
7. function stake(uint _amount) external updateReward(msg.sender)
8. function topUpPoolRewards(uint _amount) external onlyOwner updateReward(address(0))
9. function transferOwnership(address _newOwner) external onlyOwner
10. function unstake(uint _amount) external updateReward(msg.sender)
11. function unwrapBeans() external onlyOwner

CONTRACT FUNCTION

Executables

12. function updateTeamWallet(address payable _teamWallet) external onlyOwner

13. function withdrawPoolRewards(uint256 _amount) external onlyOwner updateReward(address(0))

14. function wrapBeans() external onlyOwner

CONTRACT CHECKLIST

Check	Status
Compiler errors	PASSED
Possible delays in data delivery	PASSED
Timestamp dependence	PASSED
Integer Overflow and Underflow	PASSED
Race Conditions and Reentrancy	PASSED
DoS with Revert	PASSED
DoS with block gas limit	PASSED
Methods execution permissions	PASSED
Economy model of the contract	PASSED
Private user data leaks	PASSED

CONTRACT CHECKLIST

Check	Status
Malicious Events Log	PASSED
Scoping and Declarations	PASSED
Uninitialized Storage Pointers	PASSED
Arithmetic accuracy	PASSED
Design Logic	PASSED
Impact of the exchange rate	PASSED
Oracle Calls	PASSED
Cross-function race conditions	PASSED
Fallback function security	PASSED
Safe Open Zeppelin contracts and implementation usage	PASSED

CONTRACT CHECKLIST

Check	Status
Contract correlation	UNCHECKED
Whitepaper	UNCHECKED
Website	UNCHECKED
Front Running.	UNCHECKED
Basic dApp checks	UNCHECKED

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

1. Rewards can be claimed by the user from the contract

2. The rewards have to be greater than 0 (zero)

3. If reward is $WBNB then $WBNB will be rewarded, otherwise the reward token is rewarded

```
function claimRewards() public updateReward(msg.sender) {
    uint rewards = userRewards[msg.sender];
    if (rewards > 0) {
        userRewards[msg.sender] = 0;
        userPaidRewards[msg.sender] += rewards;
        if(rewardToken == IERC20(0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c)) {
            WBNB.withdraw(rewards);
            (bool success, ) = msg.sender.call{value: rewards}("");
            require(success, "Transfer of beans failed.");
        } else
        rewardToken.safeTransfer(msg.sender, rewards);
        emit RewardPaid(msg.sender, rewards);
    }
}
```

4. Users can emergency unstake at any point of time

5. The users staked balance must be greater than 0 (zero)

6. An early unstake fee will be charged, will be sent over to the team wallet and the left over amount will be transferred to the executer of this function.

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

```
function emergencyUnstake() external updateReward(msg.sender) {
    uint amount = userStakedBalance[msg.sender];
    if(amount <= 0) revert InvalidAmount();
    userStakedBalance[msg.sender] = 0;
    _totalStaked -= amount;
    uint fee = amount * EARLY_UNSTAKE_FEE / 10000;
    uint amountDue = amount - fee;
    stakedToken.safeTransfer(teamWallet, fee);
    stakedToken.safeTransfer(msg.sender, amountDue);
    emit Unstaked(msg.sender, amount);
}
```

7. The owner of this contact can withdraw the contract's $BNB to their own address

```
function recoverBeans() external onlyOwner {
    uint balance = address(this).balance;
    (bool success, ) = msg.sender.call{value: balance}("");
    require(success, "Transfer failed.");
}
```

8. The owner of this contract withdraw any bep20 token from contract's balance to own address

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

```
function recoverWrongToken(IERC20 _token) external onlyOwner {
    uint balance = _token.balanceOf(address(this));
    _token.safeTransfer(msg.sender, balance);
}
```

9. The owner of this contract is able to change the pool duration

```
function setPoolDuration(uint _duration) external onlyOwner {
    require(poolEndTime < block.timestamp, "Pool still live");
    poolDuration = _duration;
}
```

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

10. The owner of this contract is able to update the “reward rate”, “pool start time”, “pool end time” and “update at time” values

```
function setPoolRewards(uint _amount) external onlyOwner updateReward(address(0)) {
    if (_amount <= 0) revert InvalidAmount();
    if (block.timestamp >= poolEndTime) {
        rewardRate = _amount / poolDuration;
    } else {
        uint remainingRewards = (poolEndTime - block.timestamp) * rewardRate;
        rewardRate = (_amount + remainingRewards) / poolDuration;
    }
    if(rewardRate <= 0) revert InvalidAmount();
    //if(rewardRate * poolDuration > totalRewardTokens()) revert InvalidAmount();
    poolStartTime = block.timestamp;
    poolEndTime = block.timestamp + poolDuration;
    updatedAt = block.timestamp;
}
```

11. The user can stake into the pool Staking amount must be greater than zero.

12. The staking amount must be greater than 0 (zero.)

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

```
function stake(uint _amount) external updateReward(msg.sender) {
    if(_amount <= 0) revert InvalidAmount();
    _totalStaked += _amount;
    userStakedBalance[msg.sender] += _amount;
    userUnlockedTime[msg.sender] = block.timestamp + LOCK_PERIOD;
    stakedToken.safeTransferFrom(msg.sender, address(this), _amount);
    emit Staked(msg.sender, _amount);
}
```

13. The owner of this contract can update the “reward rate" and "update at time"

```
    function topUpPoolRewards(uint _amount) external onlyOwner updateReward(address(0)) {
        uint remainingRewards = (poolEndTime - block.timestamp) * rewardRate;
        rewardRate = (_amount + remainingRewards) / poolDuration;
        if(rewardRate <= 0) revert InvalidAmount();
        // if(stakedToken == rewardToken) {
        //     if(rewardRate * poolDuration > rewardToken.balanceOf(address(this)) - _totalStaked)
revert InvalidAmount();
        // } else {
        //     if(rewardRate * poolDuration > rewardToken.balanceOf(address(this))) revert
InvalidAmount();
        // }
        updatedAt = block.timestamp;
    }
```

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

14. The owner of this contract transfers the ownership of this contract to a “_newOwner” address

```
function transferOwnership(address _newOwner) external onlyOwner {
    if(_newOwner == address(0)) revert InvalidAddress();
    owner = _newOwner;
}
```

15. Users can unstake the amount they have staked

16. The users unstake amount must be greater than zero 0 (zero)

17. The unstake time must be reached before actually unstaking

```
function unstake(uint _amount) external updateReward(msg.sender) {
    if(block.timestamp < userUnlockedTime[msg.sender]) revert TokensLocked();
    if(_amount <= 0) revert InvalidAmount();
    if(_amount > userStakedBalance[msg.sender]) revert InvalidAmount();
    _totalStaked -= _amount;
    userStakedBalance[msg.sender] -= _amount;
    stakedToken.safeTransfer(msg.sender, _amount);
    emit Unstaked(msg.sender, _amount);
}
```

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

18. The owner of this contract can wrap the contract's $BNB balance

```
function unwrapBeans() external onlyOwner {
    uint balance = WBNB.balanceOf(address(this));
    WBNB.withdraw(balance);
    assert(address(this).balance >= balance);
}
```

19. The owner of this contract can update the team wallet address

```
function updateTeamWallet(address payable _teamWallet) external onlyOwner {
    teamWallet = _teamWallet;
}
```

FUNCTIONS OF THE CONTRACT

SAFE ONE CHAIN STAKING POOL STAKING CONTRACT

20. The owner of this contract can withdraw the reward tokens from the pool rewards

```
    function withdrawPoolRewards(uint256 _amount) external onlyOwner updateReward(address(0)) {
        uint remainingRewards = (poolEndTime - block.timestamp) * rewardRate;
        rewardRate = (remainingRewards - _amount) / poolDuration;
        require(rewardRate > 0, "reward rate = 0");
        // if(stakedToken == rewardToken) {
        //     if(rewardRate * poolDuration > rewardToken.balanceOf(address(this)) - _totalStaked)
revert InvalidAmount();
        // } else {
        //     if(rewardRate * poolDuration > rewardToken.balanceOf(address(this))) revert
InvalidAmount();
        // }
        rewardToken.safeTransfer(address(msg.sender), _amount);
        updatedAt = block.timestamp;
    }
```

21. The owner of this contract can unwrap the contract's bnb

```
function wrapBeans() external onlyOwner {
    uint balance = address(this).balance;
    IWETH(WBNB).deposit{value: balance}();
    assert(IWETH(WBNB).transfer(address(this), balance));
}
```

GENERAL SUMMARY

Category	Subcategory	Result
Contract Programming	Solidity version has not specified	PASSED
	Solidity version is very aged	PASSED
	Integer overflow ~ underflow	PASSED
	Function input parameters lack checks	PASSED
	Function input parameters check bypass	PASSED
	Function access control lacking management	PASSED
	Critical operations lacking event logs	PASSED
	Human ~ contract checks bypass	PASSED
	Random number generation ~ use vulnerability	PASSED
	Fallback function misuse	PASSED

GENERAL SUMMARY

Category	Subcategory	Result
Code Specification	Race condition	PASSED
	Logical vulnerability	PASSED
	Other programming issues	PASSED
	Visibility not explicitly declared	PASSED
	Location not explicitly declared in Var. storage	PASSED
	Using keywords ~ functions to be deprecated	PASSED
	Other code specification issues	PASSED
Gas Optimization	Assert () misuse	PASSED
	High consumption ‘for ~ while’ loop	PASSED
	High consumption ‘storage’ storage	PASSED

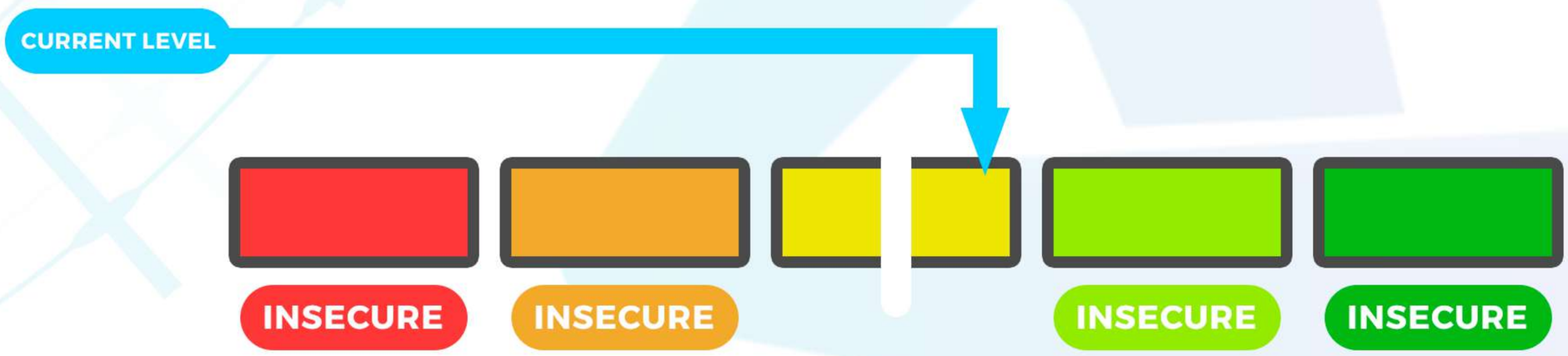
GENERAL SUMMARY

Category	Subcategory	Result
Business Risk	“Out of Gas” Attack	PASSED
	The maximum limit for mintage not set	PASSED
	“Short Address” Attack	PASSED
	“Double Spend” Attack	PASSED

EXECUTIVE SUMMARY

The $LFD solidity smart contract contains various hazards, which are highlighted here, according the usual audit assessment.

The SAFO advises concluding a further in-depth audit assessment with a reliable third party in order to get an additional assured conclusion.



We used a variety of tools, including Remix IDE and Slither among other things, to carry out our analysis.

It's crucial to remember that our conclusions are based on a careful manual audit.

All problems identified by the automatic analysis were personally examined, and any pertinent vulnerabilities were given in the General Summary section.

STATUS	CRITICAL	CRITICAL	CRITICAL	CRITICAL	CRITICAL
OPEN	0	0	0	0	0
ACKNOWLEDGED	0	0	0	0	0
RESOLVED	0	0	0	0	0

Code Quality

There is only one smart contract in the Safe One Chain Staking Pool Smart Contract protocol.

The Safe One Chain Staking Pool Smart Contract's logical algorithm is made up of the libraries that were implemented.

They are smart contracts with reusable code that, once put on the blockchain (immutable once only), is given a specific address and may be utilized repeatedly by other contracts in the protocol thanks to their properties and/or methods.

The absence of scenario and unit test scripts from the SAFO team prevents an automated evaluation of the code's integrity.

The code is generally not commented, even though doing so may give much-needed explanation for functions, return variables, and other things.

Documentation

As previously noted, incorporating comments in smart contract code is highly recommended, as they can help readers quickly comprehend the programming flow and complex code logic.

We had received a BSC explorer URL containing the Safe One Chain Staking Pool Smart Contract smart contract code.

Use of Dependencies

As previously said, adding comments to smart contract code is highly advised as they can aid users in quickly understanding complex code logic and the programming flow.

The Safe One Chain Staking Pool Smart Contract smart contract code had been sent to us in the form of a BSC explorer URL.

SEVERITY DEFINITIONS

RISK LEVEL	DESCRIPTION
Critical	Exploiting critical vulnerabilities is often an easy task that can result in token loss, amongst other consequences.
High	Whilst high-level vulnerabilities may be challenging to exploit, they can have a significant impact on smart contract execution, especially if they grant public access to critical functions.
Update At	Whilst it's important to address medium-level vulnerabilities, it's crucial to note that they cannot result in the loss of tokens.
Low	Minor vulnerabilities typically stem from unused or outdated code snippets that are unlikely to have a significant impact on execution.
-Unknown -Lowest-level vulnerabilities -Code style violations -Style and info statements can’t affect smart contracts critically	It's important to note that style and best info statements won't impact smart contracts or their functionality. These are merely practice executions and can be safely disregarded.

AUDIT FINDINGS

Critical

There were no Critical severity vulnerabilities detected.

High

3) The owner has the ability to modify various states of the contract.

One way to make it secure is to implement the governance which will vote to make any change.

Medium

1) Use of "block. timestamp": "block. timestamp" can be influenced by miners to a certain degree.

That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

2) Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

Low

There were no Low severity vulnerabilities detected

AUTOMATED TESTING

SOLIDITY STATIC ANALYSIS

SOLIDITY STATIC ANALYSIS

Select all Autorun Run

Security

Select Security

- **Transaction origin:** 'tx.origin' used
- **Check-effects-interaction:** Potential reentrancy bugs
- **Inline assembly:** Inline assembly used
- **Block timestamp:** Can be influenced by miners
- **Low level calls:** Should only be used by experienced devs
- **Block hash:** Can be influenced by miners
- **Selfdestruct:** Contracts using destructed contract can be broken

Gas & Economy

Select Gas & Economy

- **Gas costs:** Too high gas requirement of functions
- **This on local calls:** Invocation of local functions via 'this'
- **Delete dynamic array:** Use require/assert to ensure complete deletion
- **For loop over dynamic array:** Iterations depend on dynamic array's size
- **Ether transfer in loop:** Transferring Ether in a for/while/do-while loop

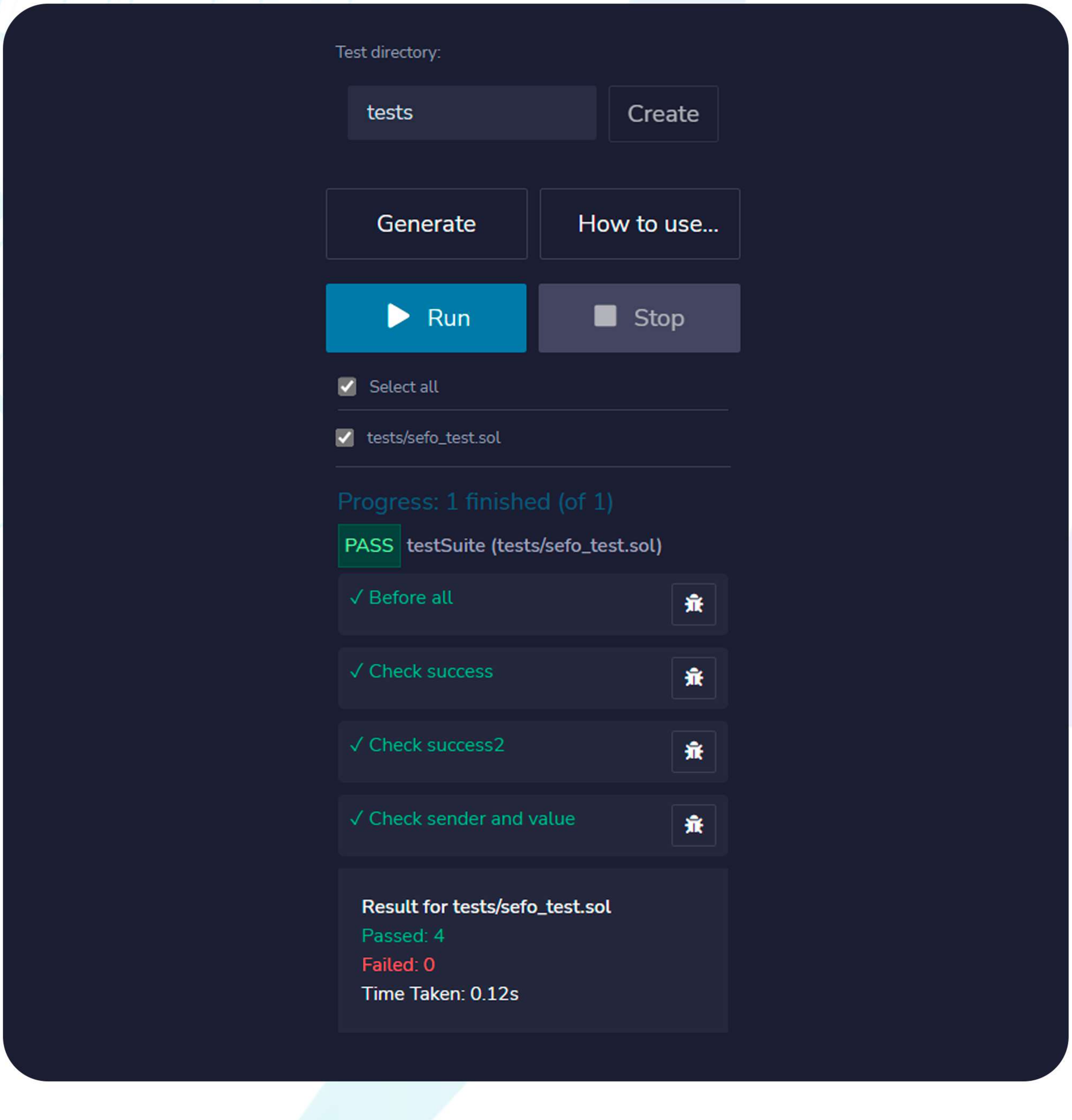
AUTOMATED TESTING

SOLIDITY STATIC ANALYSIS ~ CONTINUED

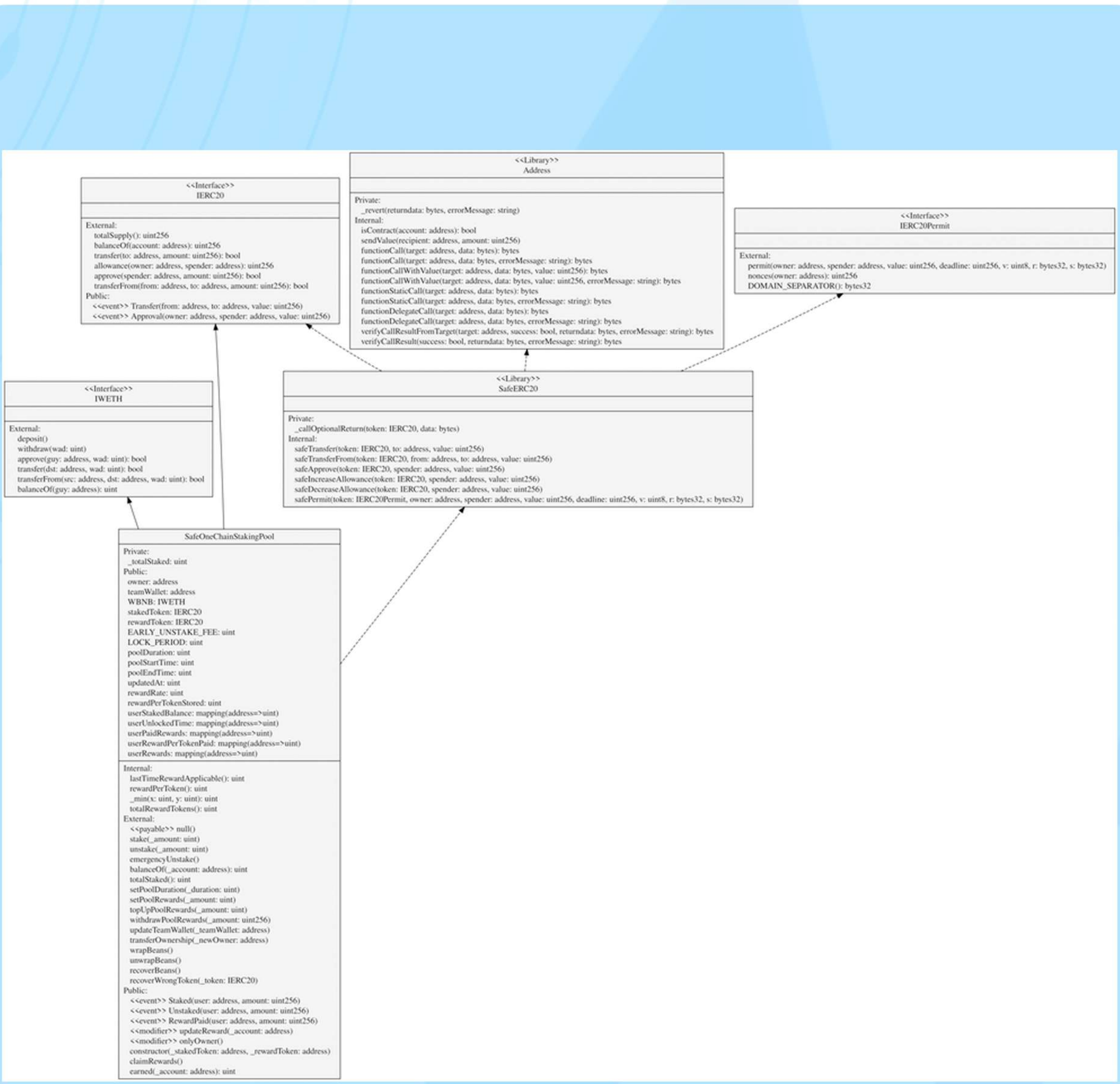
- ERC
  - Select ERC
    - ERC20:
      'decimals' should be 'uint8'
- Miscellaneous
  - Select Miscellaneous
    - Constant/View/Pure functions:
      Potentially constant/view/pure functions
    - Similar variable names:
      Variable names are too similar
    - No return:
      Function with 'returns' not returning
    - Guard conditions:
      Ensure appropriate use of require/assert
    - Result not used:
      The result of an operation not used
    - String length:
      Bytes length != String length
    - Delete from dynamic array:
      'delete' leaves a gap in array
    - Data truncated:
      Division on int/uint values truncates the result

IERC20Permit | IERC20 | SafeERC20 | Address | IWETH | SafeOneChainStakingPool

SOLIDITY UNIT TESTING



UNIFIED MODELING LANGUAGE (UML)



FUNCTIONS SIGNATURE

```
59927044 => teamWallet()
39efdf6b => EARLY_UNSTAKE_FEE()
1820cabb => LOCK_PERIOD()
8dd95002 => WBNB()
70a08231 => balanceOf(address)
372500ab => claimRewards()
008cc262 => earned(address)
7589cf2f => emergencyUnstake()
8da5cb5b => owner()
35c530dd => poolDuration()
6e271dd5 => poolEndTime()
5f96dc11 => poolStartTime()
ad08aa8f => recoverBeans()
018bcf5c => recoverWrongToken(address)
df136d65 => rewardPerTokenStored()
7b0a47ee => rewardRate()
f7c618c1 => rewardToken()
02a01dc2 => setPoolDuration(uint256)
4470c0bc => setPoolRewards(uint256)
a694fc3a => stake(uint256)
cc7a262e => stakedToken()
a9e38d36 => topUpPoolRewards(uint256)
817b1cd2 => totalStaked()
f2fde38b => transferOwnership(address)
2e17de78 => unstake(uint256)
e9ca7025 => unwrapBeans()
7cb332bb => updateTeamWallet(address)
7519ab50 => updatedAt()
c69ec430 => userPaidRewards(address)
33e630f8 => userStakedBalance(address)
0de01163 => userUnlockedTime(address)
ea6160ad => withdrawPoolRewards(uint256)
3624e890 => wrapBeans()
```

AUTOMATIC GENERAL REPORT

1- Files Description Table
2-
3- | File Name | SHA-1 Hash |
4- | Safe One Chain Staking Pool.sol|--------------|
5- Contracts Description Table
6- | Contract | Type | Bases | | |
7- |:----------:|:--------------------:|:----------------:|:----------------:|:--------------:|
8- | └ | **Function Name** | **Visibility** | **Mutability** |
9- **Modifiers** |
10- ||||||
11- | **IERC20** | Interface | |||
12- | └ | totalSupply | External ❗ | |NO❗ |
13- | └ | balanceOf | External ❗ | |NO❗ |
14- | └ | transfer | External ❗ | 🛑 |NO❗ |
15- | └ | allowance | External ❗ | |NO❗ |
16- | └ | approve | External ❗ | 🛑 |NO❗ |
17- | └ | transferFrom | External ❗ | 🛑 |NO❗ |
18- ||||||
19- | **IWETH** | Interface | |||
20- | └ | deposit| External ❗ | 💵 |NO❗ |
21- | └ | transfer| External ❗ | 🛑 |NO❗ |
22- | └ | withdraw| External ❗ | 🛑 |NO❗ |
23- | └ | transferFrom| External ❗ | 🛑 |NO❗ |
24- | └ | approve| External ❗ | 🛑 |NO❗ |
25- | └ | balanceOf | External ❗ | |NO❗ |
26- ||||||
27- | ** IERC20Permit** | Interface | |||
28- | └ | permit| External ❗ | 🛑 |NO❗ |
29- | └ | nonces | External ❗ | |NO❗ |
30- | └ | DOMAIN_SEPARATOR| External ❗ | |NO❗ |
31- ||||||
32- | ** SafeOneChainStakingPool** | Implementation | Ownable |||
33- | └ | <Constructor> | ------- | 💵 |NO❗ |
34- | └ | <Receive Ether> | external ❗ | 💵 |NO❗ |
35- | └ | stake | external ❗ | 🛑 | NO❗ |
36- | └ | unstake | External ❗ | 🛑 |NO❗ |
37- | └ | emergencyUnstake | External ❗ | 🛑 | NO❗ |
38- | └ | claimRewards| External ❗ | 🛑 |NO❗ |
Smart Contract Code Review and Security Analysis Report for
Safe One Chain Staking Pool Smart Contract
39- | └ | setPoolDuration| External ❗ | 🛑 | onlyOwner |
40- | └ | setPoolRewards| External ❗ | 🛑 | onlyOwner |
41- | └ | topUpPoolRewards| External ❗ | 🛑 | onlyOwner |
42- | └ | withdrawPoolRewards| External ❗ | 🛑 | onlyOwner |
43- | └ | updateTeamWallet| External ❗ | 🛑 | onlyOwner |
44- | └ | transferOwnership| External ❗ | 🛑 | onlyOwner |
45- | └ | wrapBeans| External ❗ | 🛑 | onlyOwner |
46- | └ | unwrapBeans| External ❗ | 🛑 | onlyOwner |
47- | └ | recoverBeans| External ❗ | 🛑 | onlyOwner |
48- | └ | recoverWrongToken| External ❗ | 🛑 | onlyOwner |
49-
50- Legend
51- | Symbol | Meaning |
52- |:--------:|:-----------|
53- | 🛑 | Function can modify state |
54- | 💵 | Function is payable

AUDIT CONCLUSION

The audit of the Smart Contract code was successful with certain caveats.

There were a few red flags raised, and SAFO used every test feasible based on the information provided and the contract code.

We cannot guarantee future results or the actual safety of the smart contract because there are an infinite number of conceivable test cases for such a comprehensive smart contract protocol.

To cover as many test cases as possible, we scanned everything using the most recent static tools as well as manual observations. Static analysis tools were used to manually inspect and evaluate the smart contracts that were under the scope.

The report's General Summary section gave a high-level explanation of the functioning of Smart Contracts.

The audit report includes all security flaws and other problems inside the examined code

SAFO METHODOLOGY

SAFO prefers to conduct its reviews in a collaborative manner and with open communication.

The security audits' objectives are to raise the caliber of the systems we examine and pursue adequate rectification to aid in user protection.

The methodology we employ in the course of our security audit procedure is as follows.

Manual Code Review:

All of the code is manually reviewed by SAFO, who search for any potential problems with code logic, error management, protocol and header parsing, cryptographic mistakes, and random number generators.

Additionally, SAFO keep an eye out for areas where improved defensive programming can lower the likelihood of errors in the future and hasten subsequent audits. We investigate dependent code and behavior when it is pertinent to a particular line of inquiry, even when our main attention is on the in-scope code.

Vulnerability Analysis:

SAFO audit methods included whitebox penetration testing, user interface interaction, and manual code analysis.

To gain a general grasp of the capabilities that the product under examination offers, SAFO look at the project's website.

SAFO then has a meeting with the programmers to understand how they see the software.

SAFO install the necessary software and use it while investigating user interactions and responsibilities.

We come up with threat models and attack surfaces while SAFO is doing this.

SAFO reviewed design documents, looked over previous audit findings, looked for related projects, looked at source code dependencies, skimmed open issue reports, and generally looked at implementation-unrelated aspects.

Documenting Results:

SAFO analyze possible security flaws and see them through to effective correction using a cautious, transparent procedure.

Even if we haven't yet confirmed the issue's viability and impact, whenever a potential problem is identified, we immediately make an Issue entry for it in this document.

This method is cautious because, even if it turns out that our suspicions weren't valid, we still recorded them early.

In general, SAFO follow a procedure that starts with unanswered questions to document the suspect, followed by code analysis, live experimentation, or automated testing to confirm the issue.

The most tentative analysis is code analysis, and SAFO tries to show test code, log captures, or pictures to show our validation. SAFO then assess the viability of an attack on an operational system.

Suggested Solutions:

SAFO looks for quick fixes that live deployments can use, and we then recommend the specifications for remediation engineering for upcoming releases.

The mitigation and remediation:

Before the specifics are made public, and after SAFO delivers its report, successful mitigation and remediation are continuing collaborative processes that should be reviewed by the developers and deployment engineers.