NOVOOS

SMART CONTRACT CODE REVIEW & SECURITY ANALYSIS REPORT FOR UCF

UCF

UC Finance

UCF

UC Finance

SMART CONTRACT CODE REVIEW & SECURITY ANALYSIS REPORT FOR UCF

TABLE OF CONTENTS

OVERVIEW

COMPLETION DATE: 8/6/2023 **LANGUAGE:** SOLIDITY

$UCF

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

Project Audited	UC Finance ERC-20 Token Smart Contract
Smart Contract Address	0xF44581d66Ff317D5D5307aa3235266F8C6694380
Contract Deployer	0xf3b78348B217a1811382655F633011E2325F4D36
Contract Owner	0xf3b78348b217a1811382655f633011e2325f4d36
Programming Language	Solidity
Blockchain	Polygon Mainnet

Novoos was requested by the UCF Smart Contract owner(s) to perform an audit on their main smart contract.

The sole purpose of the audit was to achieve the following:

- Ensure that the smart contract functions for its intended purpose.
- Potential security issues within the smart contract to be identified.

Use the reporting data in this report to analyze the risk exposure of the smart contract, and make improvements to its security within by addressing the identified issues.

UCF PROPERTIES

Smart Contract Properties

Contract Name	UC Finance
Symbol	UCF
Decimals	18
Total Supply	1,000,000 UCF
Transfer(s)	2,510
Holders	2,402 addresses
Total Distributed	1000000 UCF
Minimum Request	0.01 Matic
Deadline	1681969442 (Thursday, April 20, 2023 5:44:02 AM)
Round 1 Deadline	1679809442 (Sunday, March 26, 2023 5:44:02 AM)
Round 2 Deadline	1679809442 (Sunday, March 26, 2023 5:44:02 AM)
Smart Contract	0xF44581d66Ff317D5D5307aa3235266F8C6694380
Contract Deployer	0xf3b78348B217a1811382655F633011E2325F4D36
Contract Owner	0xf3b78348b217a1811382655f633011e2325f4d36
Blockchain Platform	Polygon Mainnet

CONTRACT FUNCTIONS

Executables

1. function add(uint256 _value) onlyOwner public
2. function approve(address _spender, uint256 _value) public returns (bool success)
3. function burn(uint256 _value) onlyOwner public
4. function DistributeAirdrop(address _participant, uint _amount) onlyOwner external
5. function DistributeAirdropMultiple(address[] _addresses, uint _amount) onlyOwner external
6. function finishDistribution() onlyOwner canDistr public returns (bool)
7. function includeInFee(address account) external onlyOwner
8. function transfer(address _to, uint256 _amount) onlyPayloadSize(2 * 32) public returns (bool success)
9. function transferFrom(address _from, address _to, uint256 _amount) onlyPayloadSize(3 * 32) public returns (bool success)
10. function transferOwnership(address newOwner) onlyOwner public
11. function updateTokensPerEth(uint _tokensPerEth) public onlyOwner
12. function withdrawAll() onlyOwner public
13. function withdraw(uint256 _wdamount) onlyOwner public
14. function withdrawForeignTokens(address _tokenContract) onlyOwner public returns (bool)

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
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

FUNCTIONS OF THE CONTRACT

UC FINANCE BEP20 TOKEN STAKING CONTRACT

1Transfers ownership of the contract to a new account (`newOwner`). Can only be called by the authorized address.

```
function transferOwnership(address newOwner) onlyOwner public {
    if (newOwner != address(0)) {
        owner = newOwner;
    }
}
```

This will transfer token for a specified address "_to" is the address to transfer. "_amount" is the amount to be transferred.

```
function transfer(address _to, uint256 _amount) onlyPayloadSize(2 * 32) public
 returns (bool success) {
    require(_to != address(0));
    require(_amount <= balances[msg.sender]);
    balances[msg.sender] = balances[msg.sender].sub(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Transfer(msg.sender, _to, _amount);
    return true;
}
```

Approve the passed address to spend the specified number of tokens on behalf of msg. sender. "spender" is the address which will spend the funds. "amount" the number of tokens to be spent.

```
function approve(address _spender, uint256 _value) public returns (bool success) {
    if (_value != 0 && allowed[msg.sender][_spender] != 0) { return false; }
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Transfer tokens from the “sender” account to the “recipient” account. The calling account must already have sufficient tokens approved for spending from the “sender” account and “sender” account must have sufficient balance to transfer. “recipient” must have sufficient allowance to transfer.

```
function transferFrom(address _from, address _to, uint256 _amount)
onlyPayloadSize(3 * 32) public
 returns (bool success) {
    require(_to != address(0));
    require(_amount <= balances[_from]);
    require(_amount <= allowed[_from][msg.sender]);
    balances[_from] = balances[_from].sub(_amount);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Transfer(_from, _to, _amount);
    return true;
}
```

Owner can distribute the tokens to “_participant” account. “_amount” should not exceed the maximum allowed distributed amount.

```
function DistributeAirdrop(address _participant, uint _amount) onlyOwner
 external {
    Distribute(_participant, _amount);
}
```

Owner can distribute the tokens to multiple “_addresses” account. “_amount” should not exceed the maximum allowed distributed amount.

```
function DistributeAirdropMultiple(address[] _addresses, uint _amount) onlyOwner
external {
    for (uint i = 0; i < _addresses.length; i++) Distribute(_addresses[i],
     _amount);
}
```

Owner can change the distribution finished state.

```
function finishDistribution() onlyOwner canDistr public returns (bool) {
    distributionFinished = true;
    emit DistrFinished();
    return true;
}
```

User can pay Matics and gets the tokens according to the rounds.

```
function getTokens() payable canDistr  public {
    uint256 tokens = 0;
    uint256 bonus = 0;
    uint256 countbonus = 0;
    uint256 bonusCond1 = 1 ether / 100;
    uint256 bonusCond2 = 1 ether / 10;
    uint256 bonusCond3 = 1 ether;
    tokens = tokensPerEth.mul(msg.value) / 1 ether;
    address investor = msg.sender;
```

Owner of this contract can change the value of tokensPerEth.

```
function updateTokensPerEth(uint _tokensPerEth) public onlyOwner {
    tokensPerEth = _tokensPerEth;
    emit TokensPerEthUpdated(_tokensPerEth);
}
```

Owner of this contract can withdraw the contract's Matic balance to own address.

```
function withdrawAll() onlyOwner public {
    address myAddress = this;
    uint256 etherBalance = myAddress.balance;
    owner.transfer(etherBalance);
}
```

Owner of this contract can burn own tokens.

```
function burn(uint256 _value) onlyOwner public {
    require(_value <= balances[msg.sender]);
    address burner = msg.sender;
    balances[burner] = balances[burner].sub(_value);
    totalSupply = totalSupply.sub(_value);
    totalDistributed = totalDistributed.sub(_value);
    emit Burn(burner, _value);
}
```

Owner of this contract can increase the total supply to use in specific purpose.

```
function add(uint256 _value) onlyOwner public {
    uint256 counter = totalSupply.add(_value);
    totalSupply = counter;
    emit Add(_value);
}
```

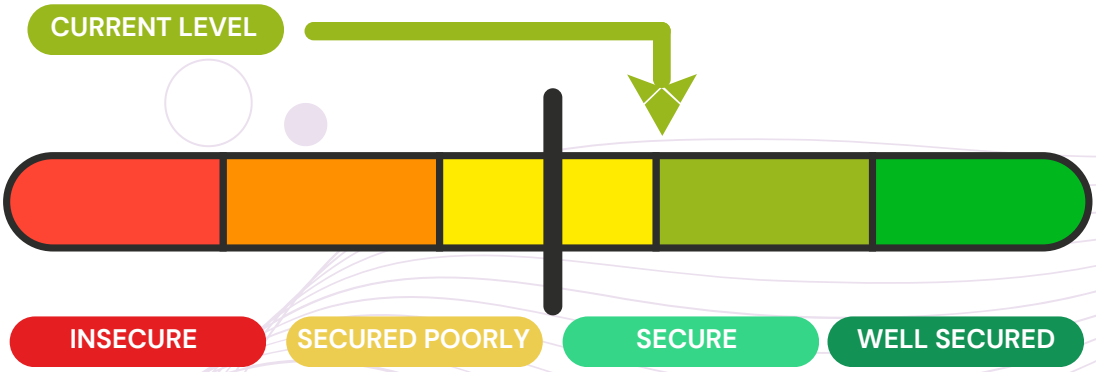
GENERAL SUMMARY

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
	Race condition	PASSED
	Logical vulnerability	PASSED
	Other programming issues	PASSED
Code Specification	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
	"Out of Gas" Attack	PASSED
Business Risk	The maximum limit for mintage not set	PASSED
	"Short Address" Attack	PASSED
	"Double Spend" Attack	PASSED

EXECUTIVE SUMMARY

According to the standard audit assessment, the **UCF** solidity smart contract has some risks as higlighted further below.

Novoos is recommending to conclude another extensive audit assesment with a reputable third-party which will provide an additional assured conclusion.



To conduct our analysis, we utilized a range of tools such as Remix IDE and Slither amongst other instruments.

However, it's important to note that our findings are based on a thorough manual audit.

We manually reviewed all issues discovered during the automated analysis and presented any relevant vulnerabilities in the **General Summary** section.

STATUS	CRITICAL	HIGH	MEDIUM	LOW	UNKNOWN
OPEN	0	0	3	0	0
ACKNOWLEDGED	0	0	0	0	0
RESOLVED	0	0	0	0	0

Code Quality

The UC Finance Smart Contract protocol consists of only one smart contract.

The libraries implemented in the UC Finance Smart Contract are part of its logical algorithm.

They are smart contracts which contain reusable code and once deployed on the blockchain (immutable ~ once only), it is assigned a specific address and its properties and/or methods can be reused many times by other contracts within the protocol.

The Novoos team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is not commented, where commenting code may provide much required documentation for functions, return variables and a lot more.

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 an explorer URL containing the UC Finance Smart Contract smart contract code.

Use of Dependencies

Based on Novoos's observations, this smart contract infrastructure employs libraries that are based on well-known industry standard open-source projects.

Additionally, the core code blocks are systematically and well-written. Furthermore, this smart contract does not interact with any external smart contracts.

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

There were no **High** severity vulnerabilities detected.

Medium

1) Block timestamp: Use of "now": "now" does not mean current time. "now" is an alias for "block.timestamp". "block.timestamp" can be influenced by miners to a certain degree, be careful.

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) The owner has the ability to modify the states of the contract. One way to make it secure is to implement the governance which will vote to make any change.

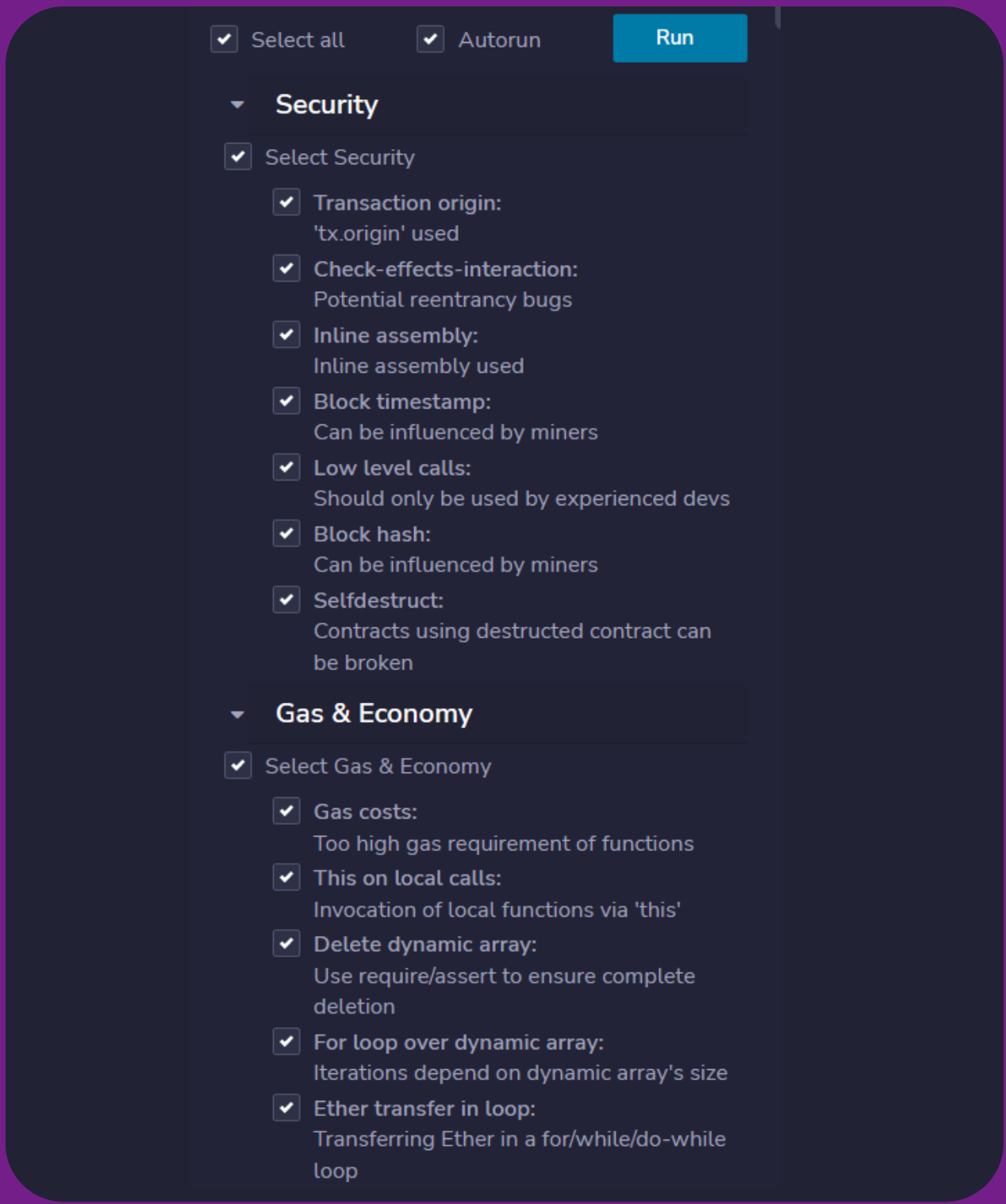
3) The contract may contain additional issuance functions, which could maybe generate a large number of tokens, resulting in significant fluctuations in token prices. It is recommended to confirm with the project team whether it complies with the token issuance instructions.

Low

There were no **Low** severity vulnerabilities detected.

AUTOMATED TESTING

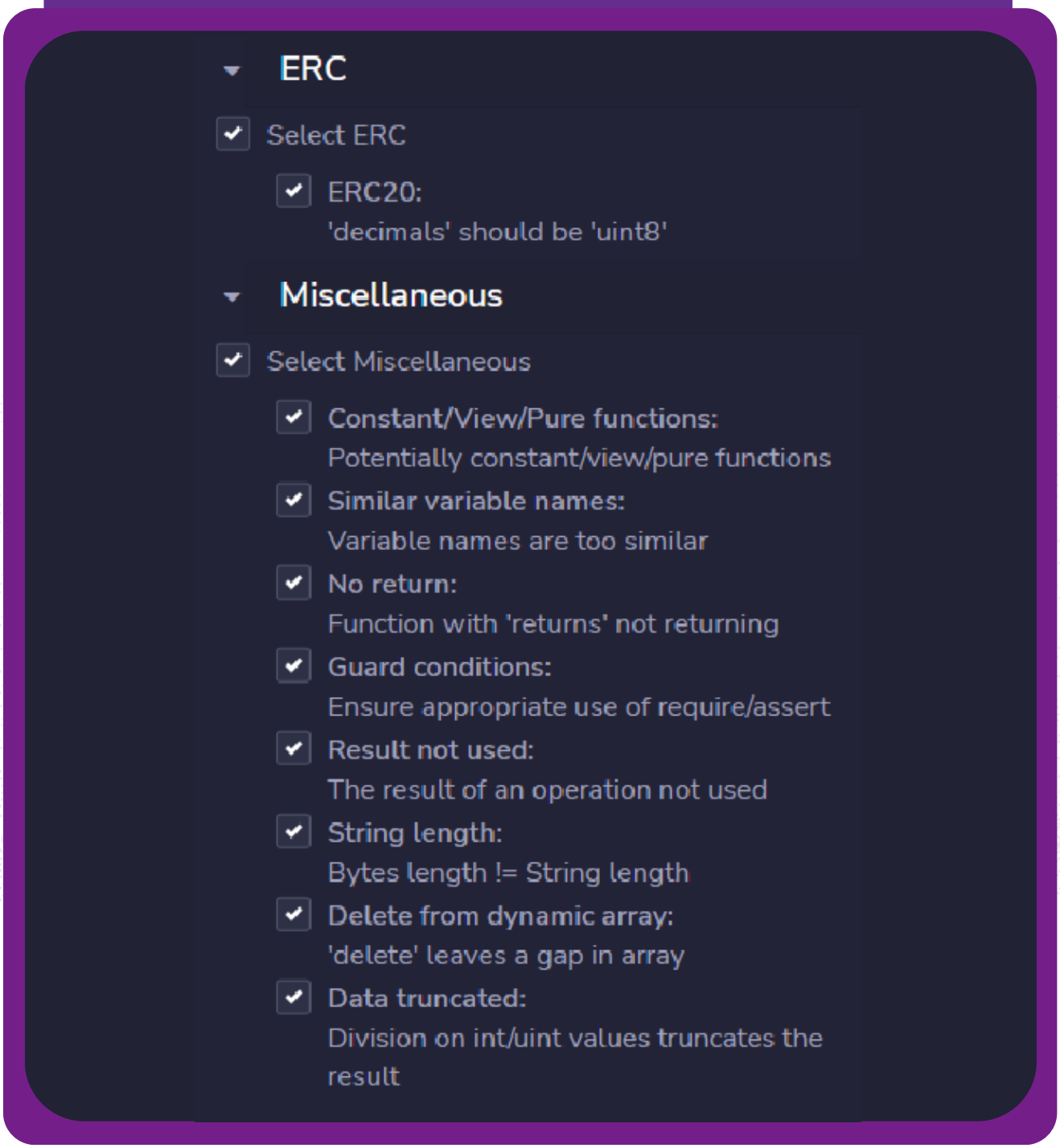
SOLIDITY STATIC ANALYSIS



AUTOMATED TESTING

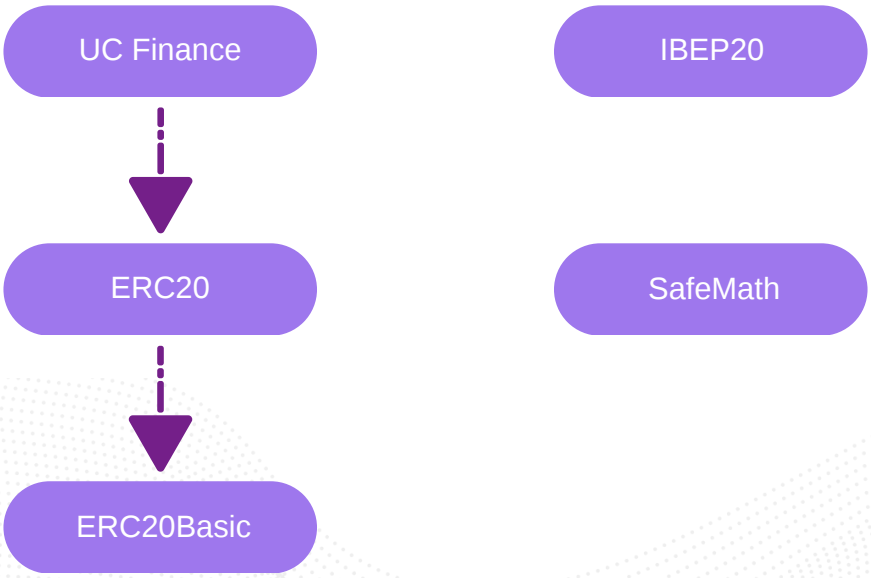
1

SOLIDITY STATIC ANALYSIS ~ CONTINUED

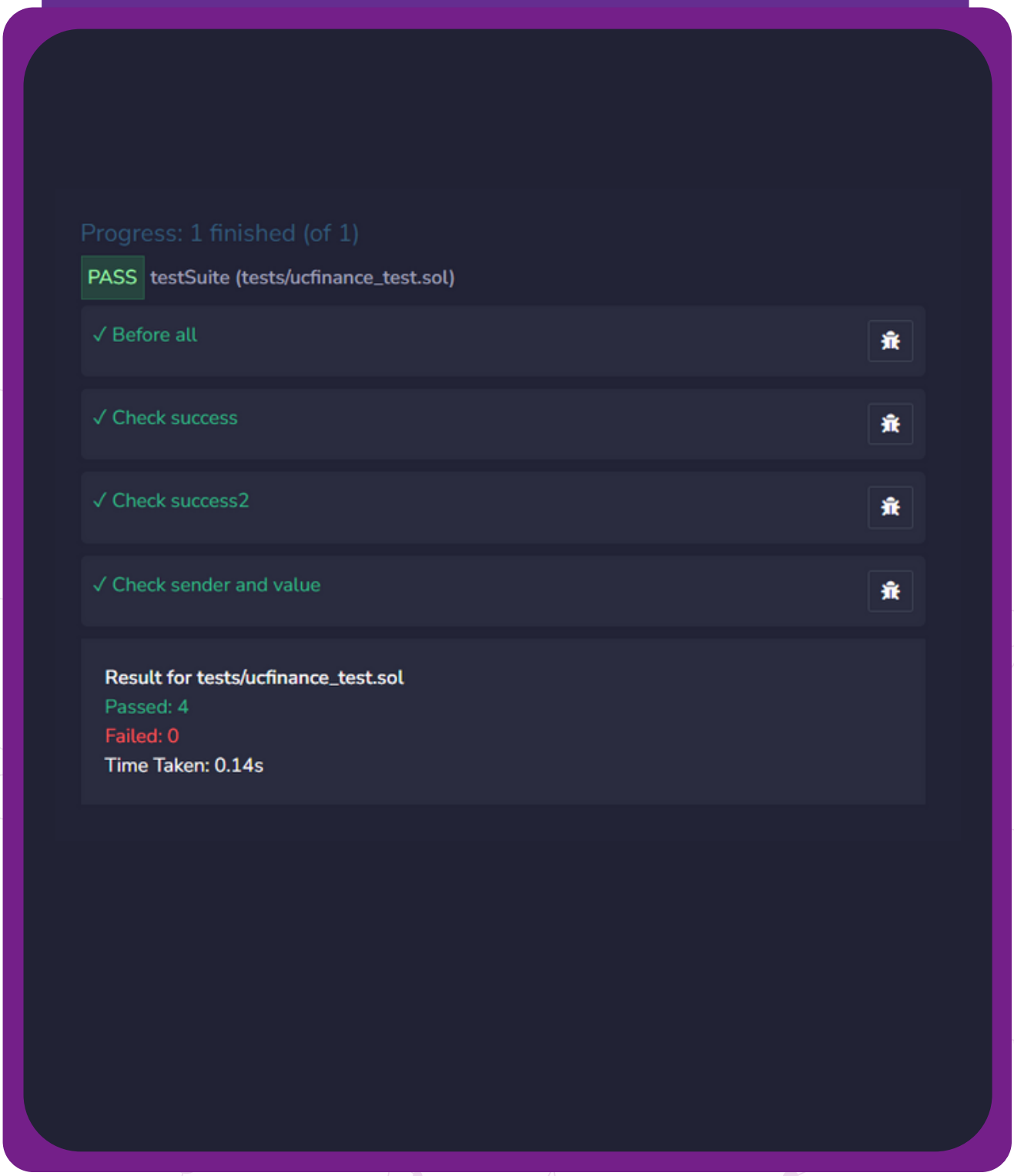


INHERITANCE GRAPH

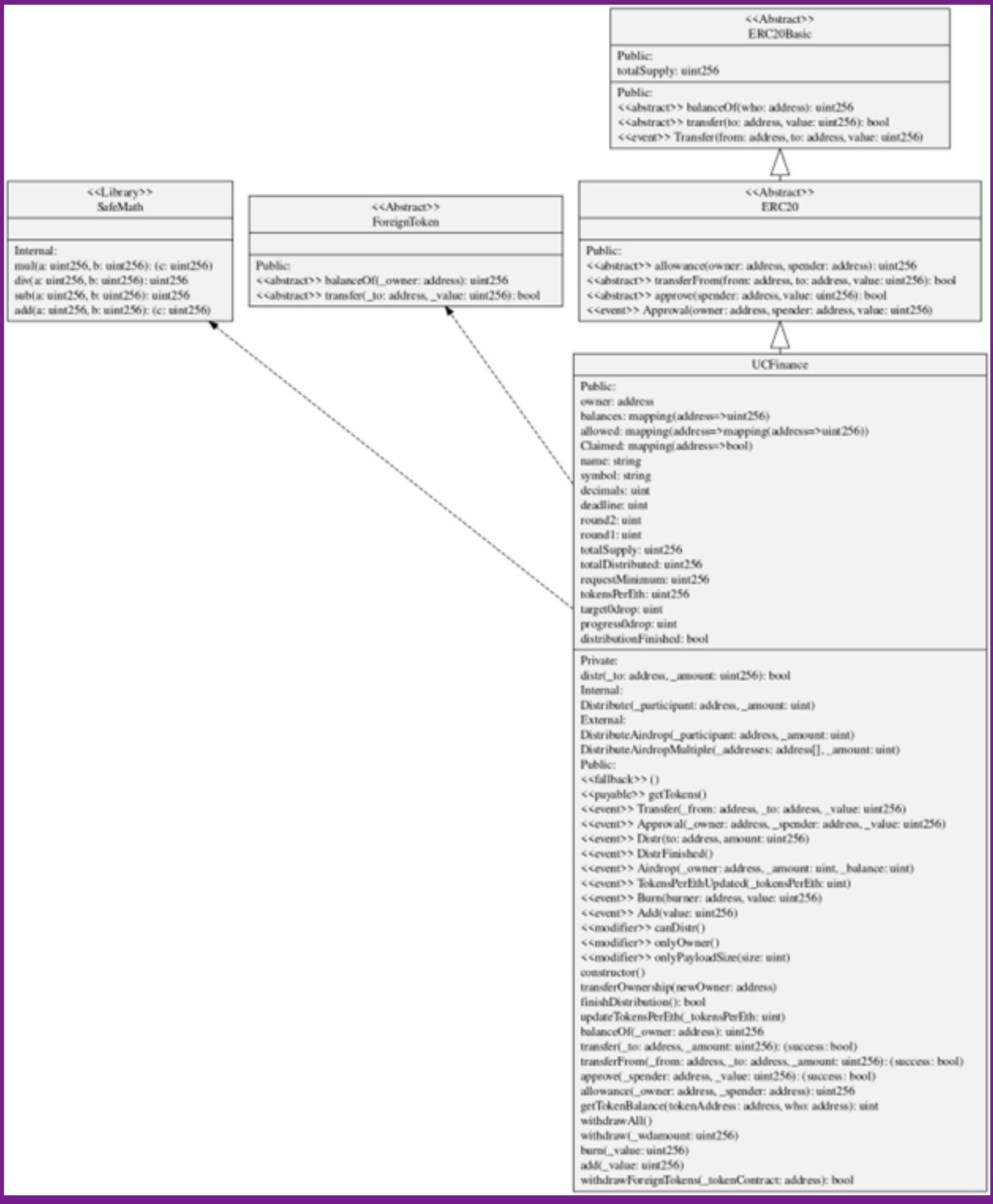
2



3 SOLIDITY UNIT TESTING



4 UNIFIED MODELING LANGUAGE (UML)



5 FUNCTIONS SIGNATURE


```
"b449c24d": "Claimed(address)",
"7809231c": "DistributeAirdrop(address,uint256)",
"f3ccb401": "DistributeAirdropMultiple(address[],uint256)",
"1003e2d2": "add(uint256)",
"dd62ed3e": "allowance(address,address)",
"095ea7b3": "approve(address,uint256)",
"70a08231": "balanceOf(address)",
"42966c68": "burn(uint256)",
"29dcb0cf": "deadline()",
"313ce567": "decimals()",
"c108d542": "distributionFinished()",
"9b1cbccc": "finishDistribution()",
"c489744b": "getTokenBalance(address,address)",
"aa6ca808": "getTokens()",
"06fdde03": "name()",
"83afd6da": "progress0drop()",
"74ff2324": "requestMinimum()",
"836e8180": "round1()",
"532b581c": "round2()",
"95d89b41": "symbol()",
"e6a092f5": "target0drop()",
"cbdd69b5": "tokensPerEth()",
"efca2eed": "totalDistributed()",
"18160ddd": "totalSupply()",
"a9059cbb": "transfer(address,uint256)",
"23b872dd": "transferFrom(address,address,uint256)",
"f2fde38b": "transferOwnership(address)",
"9ea407be": "updateTokensPerEth(uint256)",
"2e1a7d4d": "withdraw(uint256)",
"853828b6": "withdrawAll()",
"e58fc54c": "withdrawForeignTokens(address)"
```

6 AUTOMATED GENERAL REPORT – PART 1

```
Files Description Table

|  File Name  |  SHA-1 Hash  |
| UCFinance.sol|--------------|
Contracts Description Table
|  Contract  |        Type        |       Bases      |                  |             |
|:----------:|:------------------:|:----------------:|:----------------:|:-----------:|
|     └      |  **Function Name**  |  **Visibility**  |  **Mutability**  |
**Modifiers**  |
||||||
| **ERC20** | contract|  |||
| └ | Approval| Public  !  |   |NO !  |
| └ | allowance | Public  !  |   |NO !  |
| └ | approve | Public  !  | 🔴  |NO !  |
| └ | transferFrom | Public  !  | 🔴  |NO !  |
||||||
| **ERC20Basic** | contract|  |||
| └ | balanceOf| Public  !  |   |NO !  |
| └ | transfer| Public  !  | 🔴  |NO !
| **ForeignToken** | contract|  |||
| └ | balanceOf| Public  !  |   |NO !  |
| └ | transfer| Public  !  | 🔴  |NO !
||||||
| ** UC Finance ** | Implementation | Ownable |||
| └ | approve | public  !  | 🔴  |NO !  |
| └ | transfer| public  !  | 🔴  |NO !  |
| └ | transferFrom| public  !  | 🔴  |NO !  |
| └ | add| public  !  | 🔴  | onlyOwner |
| └ | burn| public  !  | 🔴  | onlyOwner |
| └ | DistributeAirdrop | External  !  | 🔴  | onlyOwner |
| └ | DistributeAirdropMultiple | External  !  | 🔴  | onlyOwner |
| └ | finishDistribution | public  !  | 🔴  | onlyOwner |
| └ | transferOwnership| public  !  | 🔴  | onlyOwner |
| └ | getTokens | public  !  | 🔴  | onlyOwner |
| └ | updateTokensPerEth | public  !  | 🔴  | onlyOwner |
| └ | withdraw | public  !  | 🔴  | onlyOwner |
| └ | withdrawAll | public  !  | 🔴  | onlyOwner |
| └ | withdrawForeignTokens | public  !  | 🔴  | onlyOwner |
| └ | allowance| Public  !  |   |NO !  |
| └ | balanceOf| Public  !  |   |NO !  |

| └ | getTokenBalance| Public  !  |   |NO !  |
Legend
|  Symbol  |  Meaning  |
|:--------:|:----------:|
|    🔴    | Function can modify state |
|    💵    | Function is payable |
```

6

AUTOMATED GENERAL REPORT – PART 2

```
60- |      └ | swapExactTokensForTokens| External ! | 🔴 |NO ! |
61- |      └ | swapTokensForExactTokens| External ! | 🔴 |NO ! |
62- |      └ | swapExactETHForTokens| External ! | 💵 |NO ! |
63- |      └ | swapTokensForExactETH| External ! | 🔴 |NO ! |
64- |      └ | swapExactTokensForETH| External ! | 🔴 |NO ! |
65- |      └ | swapETHForExactTokens| External ! | 💵 |NO ! |
66- |     |||||
67- |      ** Tomokachi ** | Implementation | Ownable |||
68- |      └ | decreaseAllowance | public ! | 🔴 | NO ! |
69- |      └ | approve | public ! | 🔴 |NO ! |
70- |      └ | increaseAllowance | public ! | 🔴 | NO ! |
71- |      └ | decreaseAllowance | public ! | 🔴 |NO ! |
72- |      └ | transfer| public ! | 🔴 |NO ! |

73- |      └ | transferFrom| public ! | 🔴 |NO ! |
74- |      └ | enableTrade | External ! | 🔴 | onlyOwner |
75- |      └ | excludeFromFee | External ! | 🔴 | onlyOwner |
76- |      └ | includeInFee | External ! | 🔴 | onlyOwner |
77- |      └ | excludeFromMaxWalletHoldingLimit | External ! | 🔴 | onlyOwner |
78- |      └ | includeInMaxWalletHoldingLimit | External ! | 🔴 | onlyOwner |
79- |      └ | transferOwnership| External ! | 🔴 | onlyOwner |
80- |      └ | UpdateMaxWalletHoldingLimit | External ! | 🔴 | onlyOwner |
81- |      └ | setNewLiquidityPair | External ! | 🔴 | onlyOwner |
82- |      └ | UpdateNoOfTokensSellToGetReward| External ! | 🔴 | onlyOwner |
83- |      └ | setSwapAndLiquifyEnabled| External ! | 🔴 | onlyOwner |
84- |      └ | UpdateWallets | External ! | 🔴 | onlyOwner |
85- |      └ | UpdateTaxFees | External ! | 🔴 | onlyOwner |
86-
87-     Legend
88- |      Symbol  |  Meaning  |
89-     |:--------:|-----------|
90- |        🔴    | Function can modify state |
91-     |    💵    | Function is payable |
```

AUDIT CONCLUSION

The Smart Contract code passed the audit successfully with some considerations to take.

There were certain warnings raised, Novoos has used all possible tests based on given information & the contract code.

Since possible test cases can be unlimited for such extensive smart contract protocol, hence we provide no such guarantee of future outcomes or the actual safety of the smart contract.

We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything. Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

Smart Contract's high-level description of functionality was presented in the **General Summary** section of the report.

The audit report contains all found security vulnerabilities and other issues within the reviewed code.

NOVOOS METHODOLOGY

Novoos prefers to work with a transparent process and make our reviews a collaborative effort.

The goals of the security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users.

The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, Novoos look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators.

Novoos also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Novoos's audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing.

Novoos look at the project's web site to get a high-level understanding of what functionality the software under review provides.

Novoos then meet with the developers to gain an appreciation of their vision of the software.

Novoos install and use the relevant software, exploring the user interactions and roles.

While Novoos does this, we brainstorm threat models and attack surfaces.

Novoos read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

Novoos follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation.

Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue.

This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities.

Novoos generally, follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests.

Code analysis is the most tentative, and Novoos strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this Novoos analyze the feasibility of an attack in a live system.

Suggested Solutions:

Novoos search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases.

The mitigation and remediation

recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after Novoos deliver our report, and before the details are made public.

TESTING SUMMARY

NOVOOS AUDIT

TELEGRAM
@novoosecosystem

EMAIL NOVOOS
audits@novoos.net

Website
WWW.NOVOOS.NET

EMAIL NOVOOS
marketing@novoos.net

BUSINESS DEVELOPMENT MANAGER
@NovoosBDM

DISCLAIMER

The Novoos team analyzed this smart contract for cybersecurity vulnerabilities and issues.
The report discloses details about the source code, compilation, deployment, and functionality.
However, the audit cannot guarantee the security of the code or its bug-free status.
This report is not a sufficient assessment of the contract's utility and safety. It is important to conduct a bug bounty program to confirm the contract's high level of security.
Whilst we have concluded this audit to the best of our ability, do not solely rely on this report.

TECHNICAL DISCLAIMER

The blockchain, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks.
Consequently, an audit of the contracts cannot guarantee their explicit security.