WHITEPAPER • PITCH DECK

NOVOOS

SMART CONTRACT CODE REVIEW & SECURITY ANALYSIS REPORT FOR TMK

TMK

TMK

SMART CONTRACT CODE REVIEW & SECURITY ANALYSIS REPORT FOR TMK

TABLE OF CONTENTS

OVERVIEW

COMPLETION DATE: 9/5/2023 LANGUAGE: SOLIDITY

$TMK

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

Project Audited	Tomokachi ERC-20 Token Smart Contract
Smart Contract Address	0xb6d54DbAc2cB10Dd97200aa4fdf9B49ae858EEEc
Programming Language	Solidity
Blockchain	ETH
Type	ERC-20

Novoos was requested by the TMK 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.

TMK PROPERTIES

Smart Contract Properties

Contract Name	Tomokachi
Symbol	TMK
Decimals	18
Total Supply	100,000,000 TMK
Transfer	1
Holders	1
# Of Tokens to add LP	100000 TMK
Max Wallet Holding	6000000 TMK
Liquidity Fee	1%
Burn Fee	1%
Team Fee	2 %
Marketing Fee	3.5%
Seed Fee	0.5 %
Total Tax Fee	8 %
Pancake Swap V2 Pair	0x065a5d3a7ee268c987383d644aef0370abc40abf
Pancake Swap V2 Router	0xd99d1c33f9fc3444f8101754abc46c52416550d1
Team Wallet	0x1be1255a0cf21dda1a85ac155e79d71558b41671
Seed Wallet	0x4E27FE7a2be6f600bb1DF7718E8a784262CA91e0
Marketing Wallet	0x7b25845a8d16fff240cdb0cf8cb27fce681b25e5
Dead Wallet	0x000000000000000000000000000000000000dead
Smart Contract	0xb6d54DbAc2cB10Dd97200aa4fdf9B49ae858EEEc
Contract Deployer	0xdb42de280c1b38eadbd5a257e9a3907104ffd6ca
Contract Owner	0xfD67d58a1cbC027Df6d795B4EBA22FE9fEc81c3b
Blockchain Platform	Binance Testnet

CONTRACT FUNCTIONS

Executables

1. function approve(address spender, uint256 amount) public virtual override returns (bool)
2. function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool)
3. function enableTrade() public onlyOwner
4. function excludeFromFee(address account) external onlyOwner
5. function excludeFromMaxWalletHoldingLimit(address account) external onlyOwner
6. function includeInMaxWalletHoldingLimit(address account) external onlyOwner
7. function includeInFee(address account) external onlyOwner
8. function renounceOwnership() public virtual onlyOwner
9. function setNewLiquidityPair(address addNewAMM, bool status) external onlyOwner
10. function setSwapAndLiquifyEnabled(bool _enabled) external onlyOwner
11. function transfer(address recipient, uint256 amount) public virtual override returns (bool)
12. function transferFrom(address sender,address recipient,uint256 amount) public virtual override returns (bool)
13. function UpdateMaxWalletHoldingLimit(uint256 maxWalletHoldingAmount) external onlyOwner
14. function UpdateNoOfTokensSellToGetReward(uint256 thresholdValue) external onlyOwner
10. function UpdateWallets(address payable newMarketingWallet,address payable newTeamWallet,address payable newSeedWallet) external onlyOwner
11. function UpdateTaxFees(uint256 newLiquidityFee,uint256 newMarketingFee,uint256 newTeamFee,uint256 newSeedFee,uint256 newBurnFee) external onlyOwner
12. function increaseAllowance(address spender, uint256 addedValue) public virtual 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
Contract correlation	UNCHECKED
Whitepaper	UNCHECKED
Website	UNCHECKED
Front Running	UNCHECKED
Basic dApp checks	UNCHECKED

FUNCTIONS OF THE CONTRACT

TOMOKACHI BEP20 TOKEN STAKING CONTRACT

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

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(
        newOwner != address(0),
        "Ownable: new owner is the zero address"
    );
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
```

2. Leaves the contract without owner. It will not be possible to call `onlyOwner` functions anymore. Can only be called by the current owner. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.

```
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
```

3. This will transfer token for a specified address “recipient” is the address to transfer. “amount” is the amount to be transferred.

```
    {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
```

4. 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 amount)
    public
    virtual
    override
    returns (bool)
{
    _approve(_msgSender(), spender, amount);
    return true;
}
```

5. 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 sender,
    address recipient,
    uint256 amount
) public virtual override returns (bool) {
    _transfer(sender, recipient, amount);
    uint256 currentAllowance = _allowances[sender][_msgSender()];
    require(
        currentAllowance >= amount,
        "IBEP20: transfer amount exceeds allowance"
    );
    return true;
}
```

6. This will increase approval number of tokens to spender address. “spender” is the address whose allowance will increase and “addedValue” are number of tokens which are going to be added in current allowance. approve should be called when allowed[spender] == 0. To increment allowed is better to use this function to avoid 2 calls (and wait until the first transaction is mined).

```
function increaseAllowance(address spender, uint256 addedValue)
    public
    virtual
    returns (bool)
{
    _approve(
        _msgSender(),
        spender,
        _allowances[_msgSender()][spender] + addedValue
    );
    return true;
}
```

7. This will decrease approval number of tokens to spender address. “spender” is the address whose allowance will decrease and “subtractedValue” are number of tokens which are going to be subtracted from current allowance.

```
function decreaseAllowance(address spender, uint256 subtractedValue)
    public
    virtual
    returns (bool)
{
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    require(
        currentAllowance >= subtractedValue,
        "IBEP20: decreased allowance below zero"
    );
    _approve(_msgSender(), spender, currentAllowance - subtractedValue);
    return true;
}
```

8. Owner of this smart contract can enable the trading.

```
function enableTrade() public onlyOwner {
    tradeEnabled = true;
}
```

9. Owner of this contract can exclude any address from fee payer lists.

```
function excludeFromFee(address account) external onlyOwner {
    _isExcludedFromFee[account] = true;
}
```

10. Owner of this contract can include any address into fee payer lists.

```
function includeInFee(address account) external onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

11. Owner of this contract can exclude any address from maximum wallet holding limit.

```
function excludeFromMaxWalletHoldingLimit(address account)
    external
    onlyOwner
{
    _isExcludedFromWalletHoldingLimit[account] = true;
}
```

12. Owner of this contract can include any address into maximum wallet holding limit addresses list.

```
function includeInMaxWalletHoldingLimit(address account)
    external
    onlyOwner
{
    require(
        account != uniswapV2Pair,
        "You can't play with Liquidity pair address"
    );
    _isExcludedFromWalletHoldingLimit[account] = false;
}
```

13. Owner of this contract can add new automatic market maker.

```
function setNewLiquidityPair(address addNewAMM, bool status)
    external
    onlyOwner
{
    _isAutomaticMarketMaker[addNewAMM] = status;
    emit AutomaticMarketMakerPairUpdated(addNewAMM, status);
}
```

14. Owner of this contract can enable/disable the swap and liquify condition.

```
function setSwapAndLiquifyEnabled(bool _enabled) external onlyOwner {
    swapAndLiquifyEnabled = _enabled;
    emit SwapAndLiquifyEnabledUpdated(_enabled);
}
```

15. Owner of this contract can update the maximum wallet holding limit, but new value must be greater or equal than 1 million tokens.

```
function UpdateMaxWalletHoldingLimit(uint256 maxWalletHoldingAmount)
    external
    onlyOwner
{
    require(
        maxWalletHoldingAmount * 10**_decimals >= 1_000_000 * 10**_decimals,
        "Amount should be greater or equal to 1 Millin Tokens"
    );
    _maxWalletHoldingLimit = maxWalletHoldingAmount * 10**_decimals;
    emit MaxWalletHoldingAmountUpdated(_maxWalletHoldingLimit);
}
```

16. Owner of this contract can update the marketing wallet, team wallet, seed wallet, but new addresses should not be dead addresses.

```
function UpdateWallets(
    address payable newMarketingWallet,
    address payable newTeamWallet,
    address payable newSeedWallet
) external onlyOwner {
    require(
        newMarketingWallet != address(0) &&
            newTeamWallet != address(0) &&
            newSeedWallet != address(0),
        "You can't set zero address"
    );
    MarketingWalletAddress = newMarketingWallet;
    TeamWalletAddress = newTeamWallet;
    SeedWalletAddress = newSeedWallet;
}
```

17. Owner of this contract can update the liquidity fee, marketing fee, team fee, seed fee and burn fee but the total of all fesses should not be greater than 8 %.

```
function UpdateTaxFees(
    uint256 newLiquidityFee,
    uint256 newMarketingFee,
    uint256 newTeamFee,
    uint256 newSeedFee,
    uint256 newBurnFee
) external onlyOwner {
    require(
        newLiquidityFee +
            newMarketingFee +
            newTeamFee +
            newBurnFee +
            newSeedFee <=
            800,
        "you can't set more than 8%"
    );
    _liquidityFee = newLiquidityFee;
    _MarketingFee = newMarketingFee;
    _TeamFee = newTeamFee;
    _BurnFee = newBurnFee;
    _SeedFee = newSeedFee;
    TotalTaxFee =
        _liquidityFee +
        _BurnFee +
        _TeamFee +
        _MarketingFee +
        _SeedFee;
    emit TaxFeeUpdated(TotalTaxFee);
}
```

18. Owner of this contract can update the number of tokens to sell to add to liquidity.

```
function UpdateNoOfTokensSellToGetReward(uint256 thresholdValue)
    external
    onlyOwner
{
    numTokensSellToAddToLiquidity = thresholdValue * 10**_decimals;
    emit MinTokensBeforeSwapUpdated(numTokensSellToAddToLiquidity);
}
```

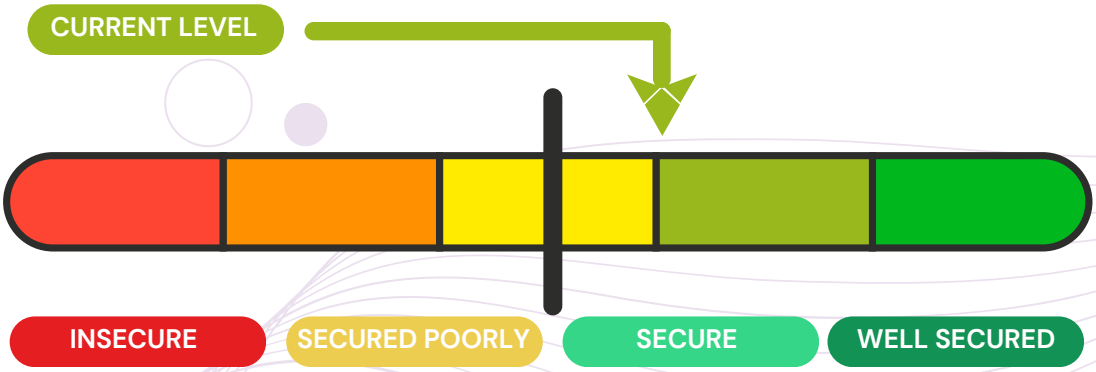
GENERAL SUMMARY

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 **TMK** 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	2	0	0
ACKNOWLEDGED	0	0	0	0	0
RESOLVED	0	0	0	0	0

Code Quality

The Tomokachi Smart Contract protocol consists of only one smart contract.

The libraries implemented in the Tomokachi 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 a ETH explorer URL containing the Tomokachi 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) 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) The owner has the ability to modify the states of the contract.

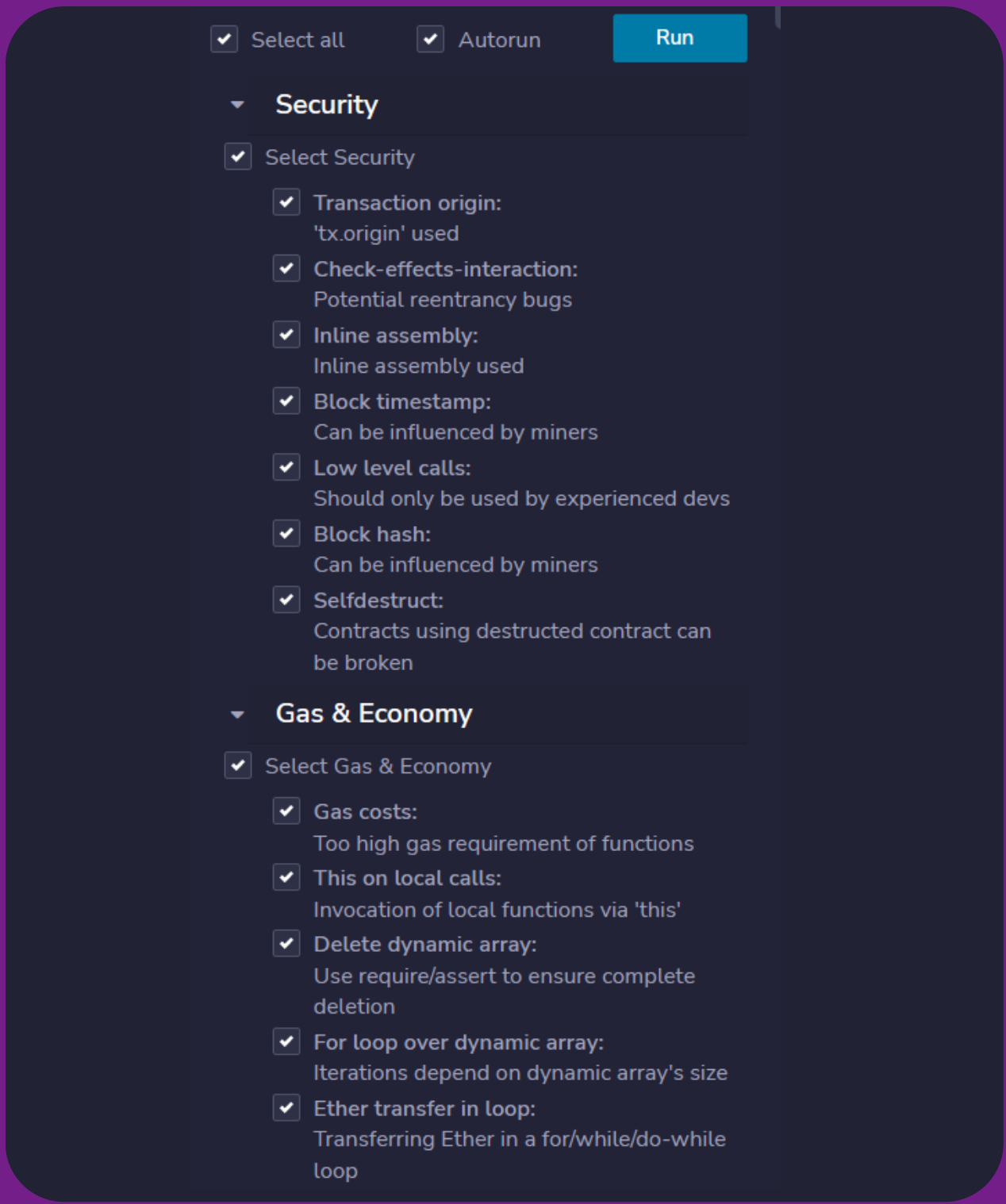
One way to make it secure is to implement the governance which will vote to make any change.

Low

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

AUTOMATED TESTING

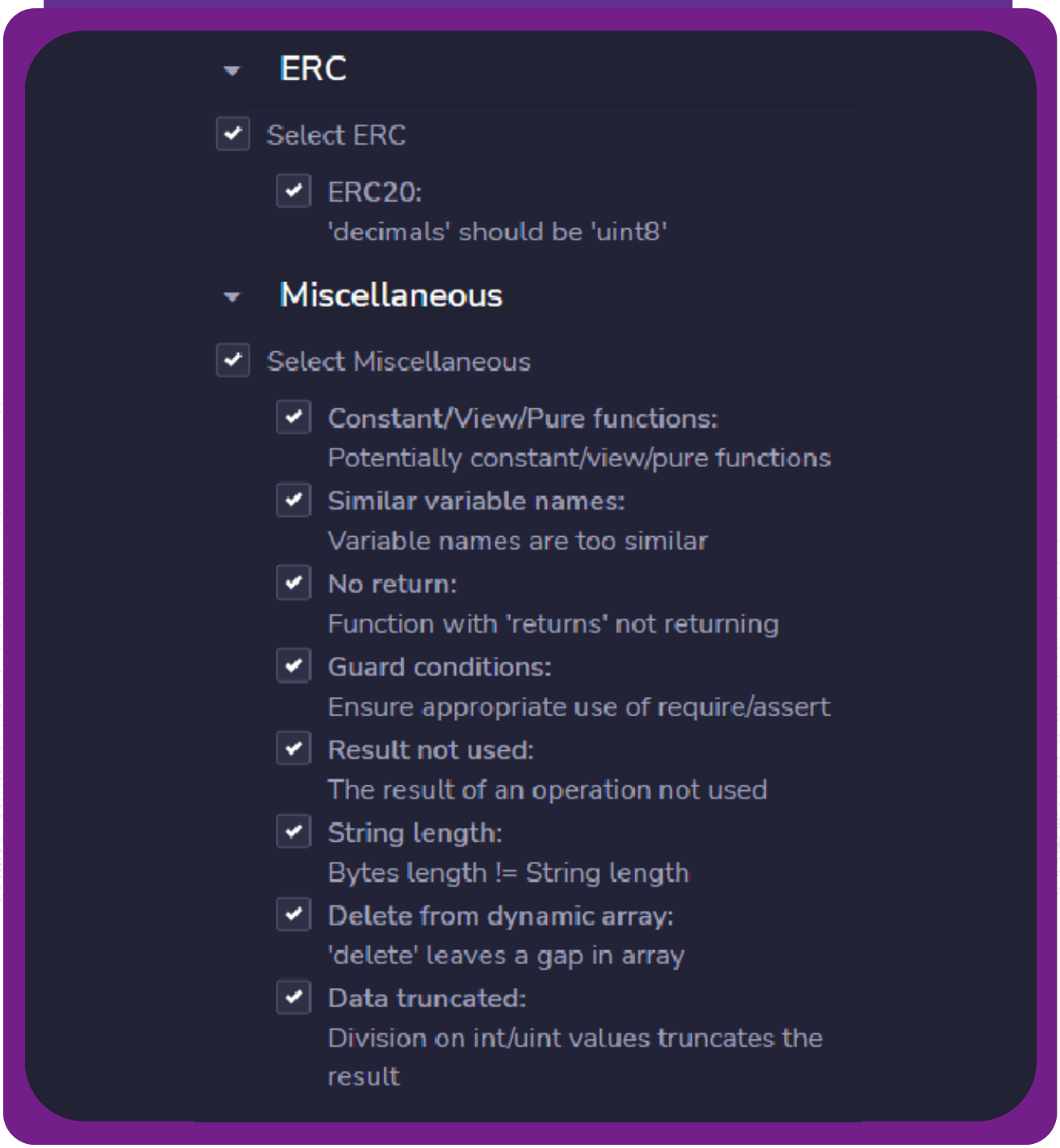
SOLIDITY STATIC ANALYSIS



AUTOMATED TESTING

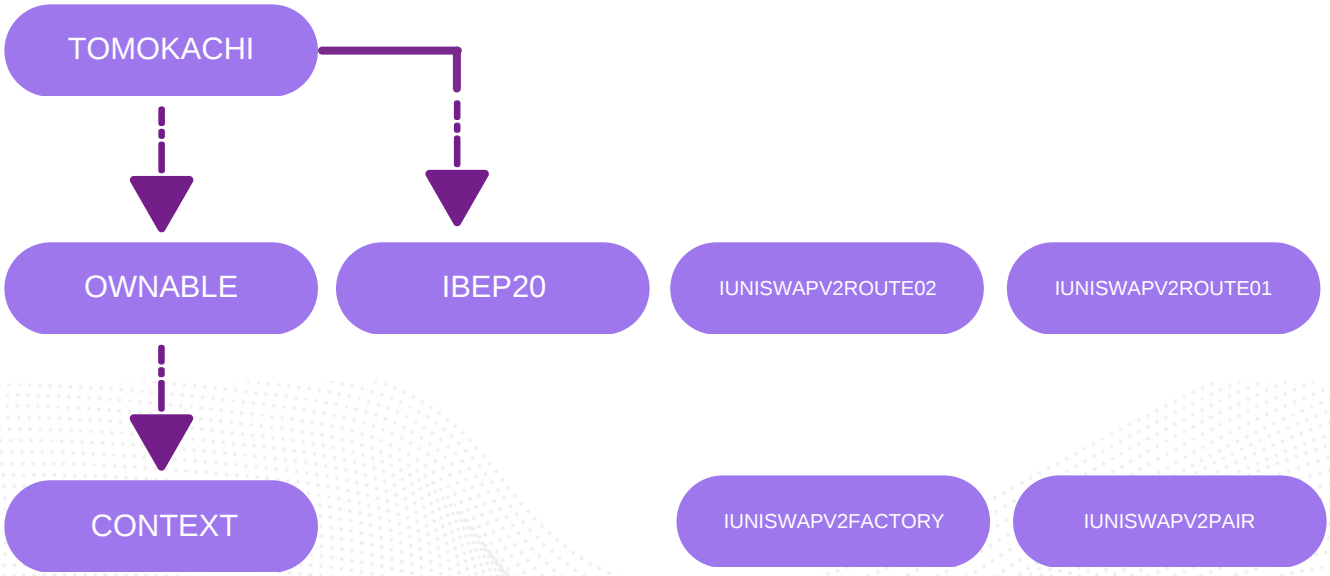
1

SOLIDITY STATIC ANALYSIS ~ CONTINUED

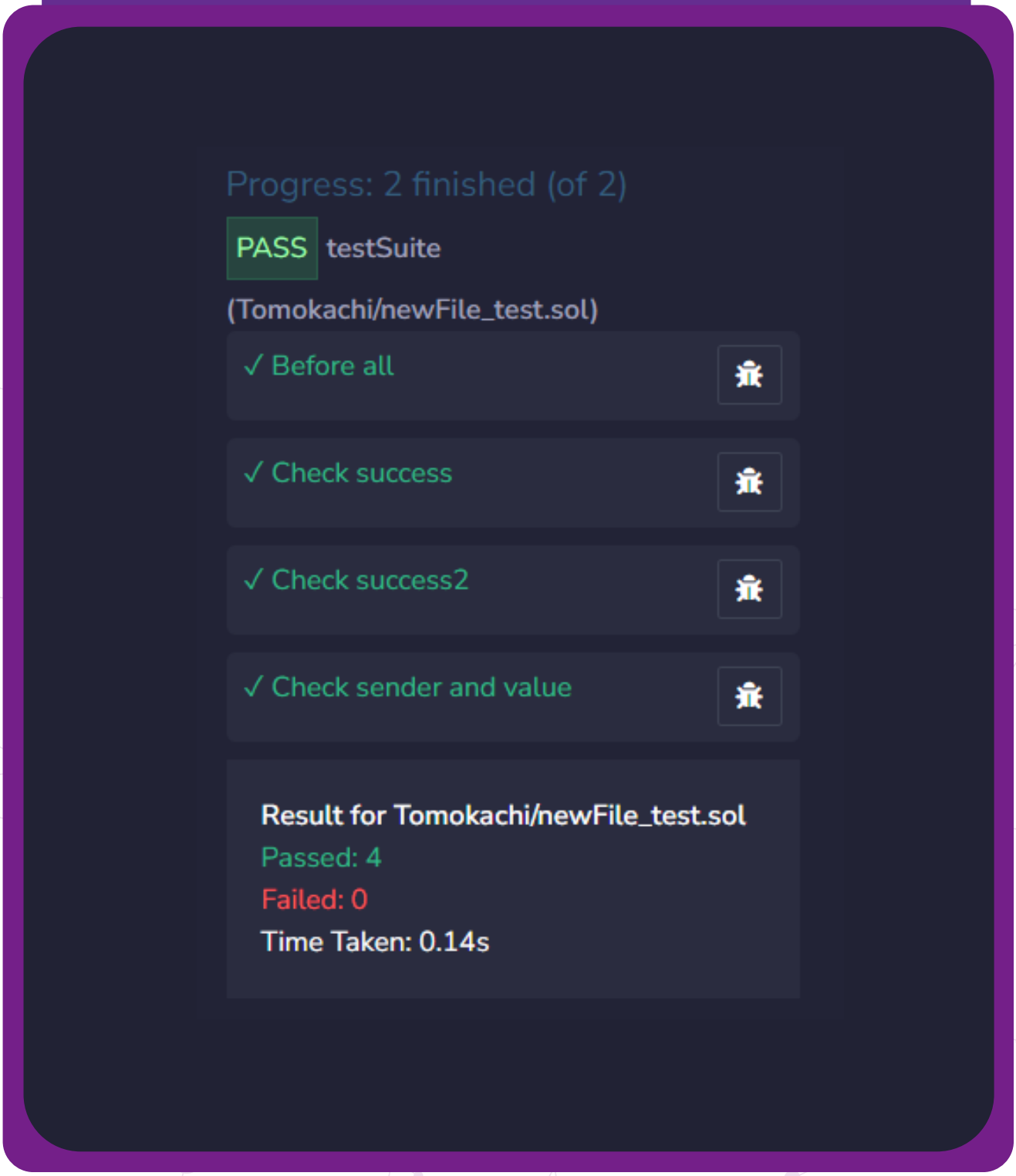


INHERITANCE GRAPH

2



3 SOLIDITY UNIT TESTING



4

UNIFIED MODELING LANGUAGE (UML)

5 FUNCTIONS SIGNATURE

```
{
    "39509351": "increaseAllowance(address,uint256)",
    "0a6a7548": "DeadWalletAddress()",
    "39e6cd66": "MarketingWalletAddress()",
    "f76c1410": "SeedWalletAddress()",
    "00712e17": "TeamWalletAddress()",
    "70f58c37": "TotalTaxFee()",
    "be9e18a4": "UpdateMaxWalletHoldingLimit(uint256)",
    "9429b9fe": "UpdateNoOfTokensSellToGetReward(uint256)",
    "7cb2bf82": "UpdateTaxFees(uint256,uint256,uint256,uint256,uint256)",
    "0dd99176": "UpdateWallets(address,address,address)",
    "32424aa3": "_decimals()",
    "124035a8": "_maxWalletHoldingLimit()",
    "3eaaf86b": "_totalSupply()",
    "dd62ed3e": "allowance(address,address)",
    "095ea7b3": "approve(address,uint256)",
    "70a08231": "balanceOf(address)",
    "313ce567": "decimals()",
    "a457c2d7": "decreaseAllowance(address,uint256)",
    "0099d386": "enableTrade()",
    "437823ec": "excludeFromFee(address)",
    "70035ba5": "excludeFromMaxWalletHoldingLimit(address)",
    "ea2f0b37": "includeInFee(address)",
    "b7a9a0af": "includeInMaxWalletHoldingLimit(address)",
    "3979e9ca": "isAutomaticMarketMaker(address)",
    "cb4ca631": "isExcludedFromTax(address)",
    "b40f9469": "isExcludedFromWalletLimit(address)",
    "06fdde03": "name()",
    "d12a7688": "numTokensSellToAddToLiquidity()",
    "8da5cb5b": "owner()",
    "715018a6": "renounceOwnership()",
    "7d3ddc92": "setNewLiquidityPair(address,bool)",
    "c49b9a80": "setSwapAndLiquifyEnabled(bool)",
    "4a74bb02": "swapAndLiquifyEnabled()",
    "95d89b41": "symbol()",
    "18160ddd": "totalSupply()",
    "d621e813": "tradeEnabled()",
    "a9059cbb": "transfer(address,uint256)",
    "23b872dd": "transferFrom(address,address,uint256)",
    "f2fde38b": "transferOwnership(address)",
    "49bd5a5e": "uniswapV2Pair()",
    "1694505e": "uniswapV2Router()"
}
```

6

AUTOMATED GENERAL REPORT – PART 1

```
Files Description Table

|      File Name      |  SHA-1 Hash  |
| Tomokachi.sol|--------------|
Contracts Description Table
|      Contract       |         Type         |       Bases       |                   |              |
|:--------------------:|:--------------------:|:-----------------:|:-----------------:|:-------------:|
|          └          |  **Function Name**   |  **Visibility**   |  **Mutability**   |
**Modifiers**  |
|  |||||
|  **IBEP20** | Interface |  |||
|  └ | totalSupply | External  ! |  |NO ! |
|  └ | balanceOf | External  ! |  |NO ! |
|  └ | transfer | External  ! |  🔴  |NO ! |
|  └ | allowance | External  ! |  |NO ! |
|  └ | approve | External  ! |  🔴  |NO ! |
|  └ | transferFrom | External  ! |  🔴  |NO ! |
|  |||||
|  **IPancakePair** | Interface |  |||
|  └ | totalSupply | External  ! |  |NO ! |
|  └ | decimals | External  ! |  |NO ! |
|  └ | symbol | External  ! |  |NO ! |
|  └ | name | External  ! |  |NO ! |
|  └ | balanceOf | External  ! |  |NO ! |
|  └ | nonces| External  ! |  |NO ! |
|  └ | PERMIT_TYPEHASH| External  ! |  |NO ! |
|  └ | DOMAIN_SEPARATOR| External  ! |  |NO ! |
|  └ | transfer | External  ! |  🔴  |NO ! |
|  └ | allowance | External  ! |  |NO ! |
|  └ | approve | External  ! |  🔴  |NO ! |
|  └ | transferFrom | External  ! |  🔴  |NO ! |
|  └ | permit| External  ! |  🔴  |NO ! |
|  └ | MINIMUM_LIQUIDITY| External  ! |  |NO ! |
|  └ | factory| External  ! |  |NO ! |
|  └ | token0| External  ! |  |NO ! |
|  └ | token1| External  ! |  |NO ! |
|  └ | getReserves| External  ! |  |NO ! |
|  └ | price0CumulativeLast| External  ! |  |NO ! |
|  └ | price1CumulativeLast| External  ! |  |NO ! |
|  └ | kLast| External  ! |  |NO ! |
|  └ | mint| External  ! |  🔴  |NO ! |
|  └ | burn| External  ! |  🔴  |NO ! |
|  └ | swap | External  ! |  🔴  |NO ! |
|  └ | skim | External  ! |  🔴  |NO ! |
|  └ | sync| External  ! |  🔴  |NO ! |
|  └ | initialize| External  ! |  🔴  |NO ! |
|  |||||
|  **IPancakeRouter01** | Interface |  |||
|  └ | quote| External  ! |  |NO ! |
|  └ | getAmountIn| External  ! |  |NO ! |
|  └ | getAmountOut| External  ! |  |NO ! |
|  └ | getAmountsOut| External  ! |  |NO ! |
|  └ | getAmountsIn| External  ! |  |NO ! |
|  └ | addLiquidity| External  ! |  🔴  |NO ! |
|  └ | addLiquidityETH | External  ! |  💵  |NO ! |
|  └ | removeLiquidity | External  ! |  🔴  |NO ! |
|  └ | removeLiquidityETH| External  ! |  🔴  |NO ! |
|  └ | removeLiquidityWithPermit| External  ! |  🔴  |NO ! |
|  └ | removeLiquidityETHWithPermit| External  ! |  🔴  |NO ! |
```

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.