its__msn itsmsn

# ITISH

AUDIT COMPANY

# Audit Details

**Contract Name**
MissInternetToken

**Deployer address**

0x8B64154c6bd720fb9a0E6184Bd67e9B19aD18193

**Client contacts:**

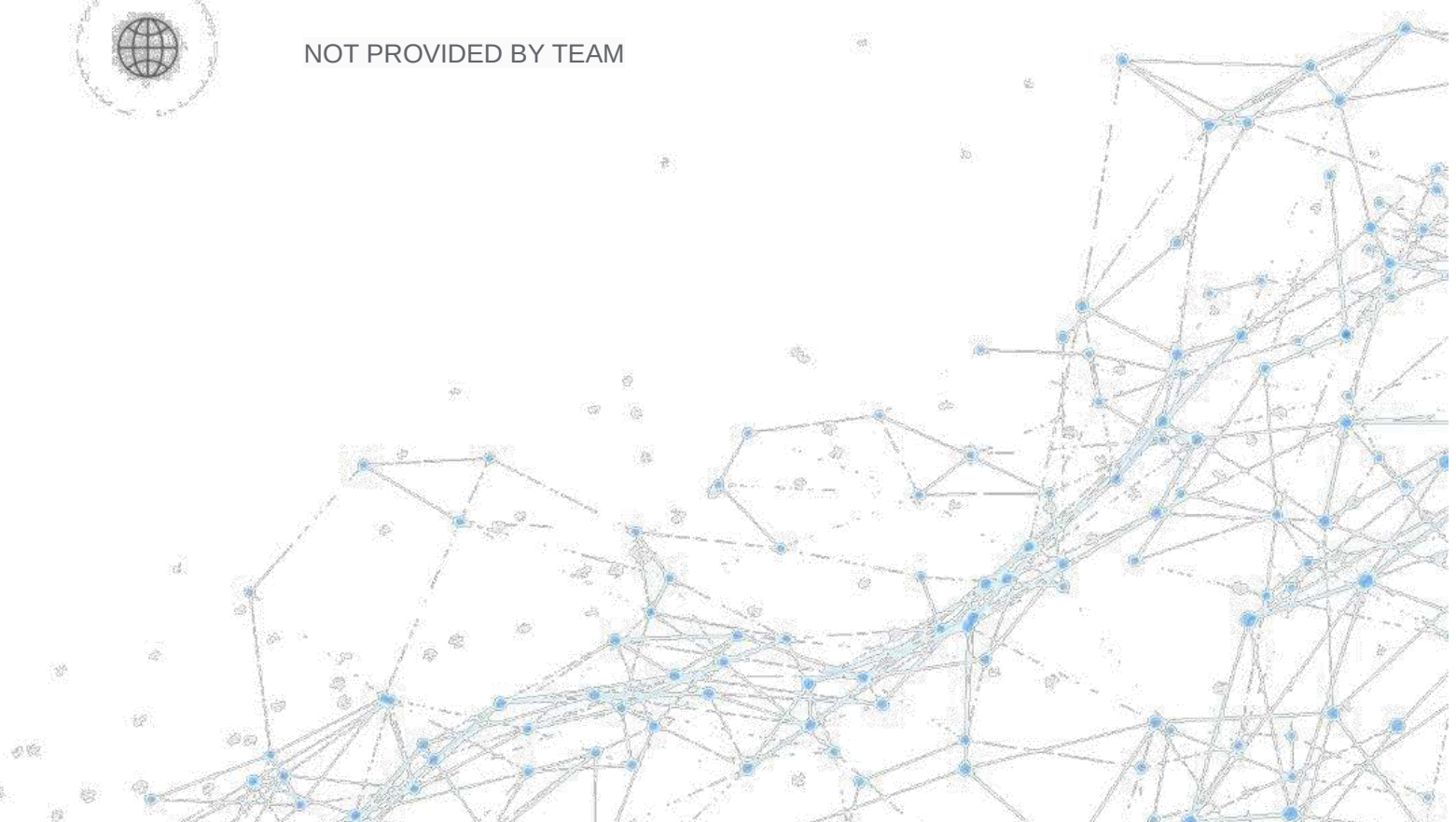
MissInternetToken
team

**Blockchain**

Binance

**Project website:**

NOT PROVIDED BY TEAM



# Disclaimer

**This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.**

**DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Itish and its affiliates (includingholding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Itish) owe no duty of care towards you or any other person, nor does Itish make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Itish hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Itish hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Itish, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.**

**The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.**

# Background

**Itish was commissioned Miss internet token to perform an audit of smart contracts:**

**https://bscscan.com/token/0x8B64154c6bd720fb9a0E6184Bd67e9B19aD18193**

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

- **Ensure that the smart contract functions as intended.**
- **Identify potential security issues with the smart contract.**

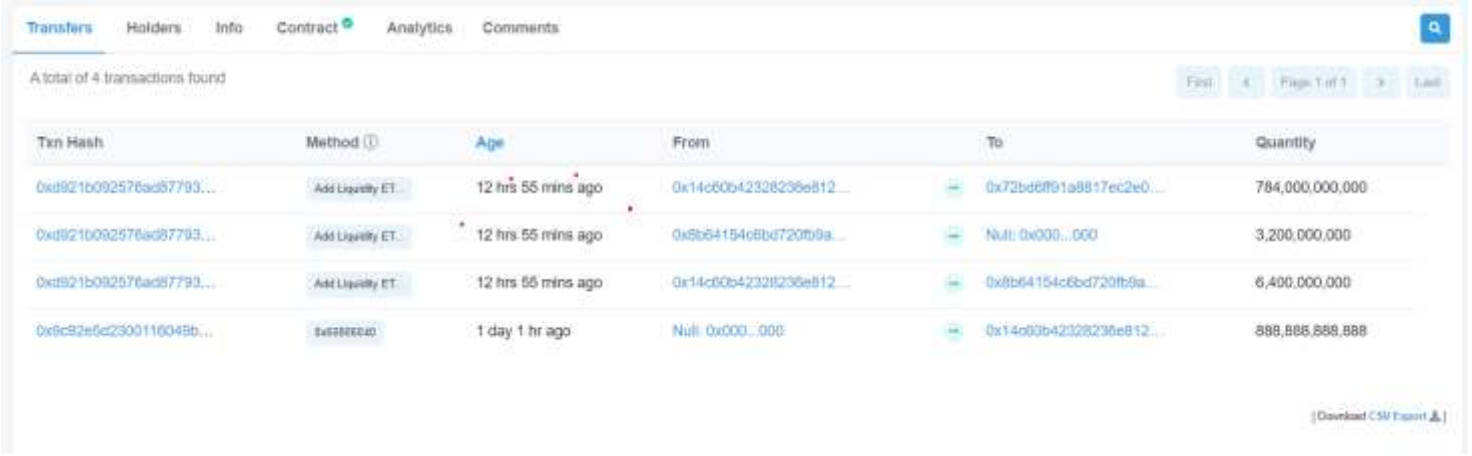
**The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.**

# Contract Details

## Token contract details for 31.03.2023

| | |
|---|---|
| **contract name** | **MissInternetToken** |
| Contract creator | 0x8B64154c6bd720fb9a0E6184Bd67e9B19aD18193 |
| **Transaction’s count** | **4** |

# Contract TopTransactions



## Token Functions Details

```
Owner()
Check_owner()
renounceOwnership()
transferOwnership()
nonReentrantBefore()
nonReentrantafter()
name()
symbol()
decimal()
total supply()
balanceof()
transfer()
allowance()
approve()
transferfrom()
increase_allownace()
decrease_allownace()
burn()
mint()
spendallownace()
aftertokentransfer()
burnfrom()
```

## Contract Interface Details

```
interface IERC20
interface IERC20Metadata is IERC20
```

# Issues Checking Status

| Issue description | checking status |
|---|---|
| 1. Compiler errors. | Passed |
| 2. Compiler Compatibilities | Passed |
| 3. Possible delays in data delivery. | Passed |
| 4. Oracle calls. | Moderate |
| 5. Front running. | Passed |
| 6. Timestamp dependence. | Passed |
| 7. Integer Overflow and Underflow. | Passed |
| 8. DoS with Revert. | Severe |
| 9. DoS with block gas limit. | Moderate |
| 10. Methods execution permissions. | Passed |
| 11. Economy model of the contract. | Passed |
| 12. The impact of the exchange rate on the logic. | Severe |
| 13. Private user data leaks. | Passed |
| 14. Malicious Event log. | Passed |
| 15. Scoping and Declarations. | Passed |
| 16. Uninitialized storage pointers. | Passed |
| 17. Arithmetic accuracy. | poor |
| 18. Design Logic. | poor |

| | | |
|---|---|---|
| 19. | **Cross-function race conditions.** | **Passed** |
| 20. | Safe Open Zeppelin contracts implementation and usage. | pass |
| 21. | **Fallback function security.** | **Passed** |

# Security Issues

## High Security Issues

### Issue # 1:

**INCORRECT ACCESS CONTROL**

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

```
 * @dev See {IERC20-totalSupply}.
 */
function totalSupply() public view virtual override returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account) public view virtual override returns (uint256) {
    return _balances[account];
}

/**
 * @dev See {IERC20-transfer}.
 *
 * Requirements:
 *
 * - `to` cannot be the zero address.
```

### Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

### Type # 2:

### Issue # 1:

**INCORRECT ACCESS CONTROL**

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

```
function transferFrom(address from, address to, uint256 amount) public virtual override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    transfer(to, amount);
    return true;
}
```

## Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

## Type # 3:

## Issue # 1:

| **INCORRECT ACCESS CONTROL**<br><br>Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. |
|---|

## Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

## Low Severity Issues

### Issue # 1:

**MISSING EVENTS**

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract `Miss internet token` was found to be missing these events on the function `_transfer` which would make it difficult or impossible to track these transactions off-chain.

```
 * Requirements:
 *
 * - `to` cannot be the zero address.
 * - the caller must have a balance of at least `amount`.
 */
function transfer(address to, uint256 amount) public virtual override returns (bool) {
    address owner = _msgSender();
    _transfer(owner, to, amount);
    return true;
}

/**
 * @dev See {IERC20-allowance}.
 */
function allowance(address owner, address spender) public view virtual override returns (uint256) {
    return _allowances[owner][spender];
}

/**
 * @dev See {IERC20-approve}.
```

### Remediation # 1:

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

### Issue # 2:

**MISSING EVENTS**

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract `MISS EVENT TOKEN` was found to be missing these events on the function `_mint` which would make it difficult or impossible to track these transactions off-chain.

```
56      * Emits a {Transfer} event with `from` set to the zero address.
57      *
58      * Requirements:
59      *
60      * - `account` cannot be the zero address.
61      */
62     function _mint(address account, uint256 amount) internal virtual {
63         require(account != address(0), "ERC20: mint to the zero address");
64
65         _beforeTokenTransfer(address(0), account, amount);
66
67         _totalSupply += amount;
68         unchecked {
69             // Overflow not possible: balance + amount is at most totalSupply + amount, which is checked above.
70             _balances[account] += amount;
71         }
72         emit Transfer(address(0), account, amount);
73
74         _afterTokenTransfer(address(0), account, amount);
```

## Remediation # 1:

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

## Issue # 3:

**MISSING EVENTS**

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract `Miss event token` was found to be missing these events on the function `_burn` which would make it difficult or impossible to track these transactions off-chain.

```
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBalance <= totalSupply.
            _totalSupply -= amount;
        }
```

## Remediation # 1:

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

## TYPE 2:
## Issue # 1:

**OUTDATED COMPILER VERSION**

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

```
1  // SPDX-License-Identifier: MIT
2  // OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)
3
4  pragma solidity ^0.8.0;
5
6  import "../utils/Context.sol";
7
8  /**
9   * @dev Contract module which provides a basic access control mechanism, where
10  * there is an account (an owner) that can be granted exclusive access to
11  * specific functions.
```

## Remediation # 1:

It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well.
Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks.
Consider using the solidity version, which patches most solidity vulnerabilities.

## TYPE 3:
## Issue # 1:

**USE OF FLOATING PRAGMA**

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.

## Remediation # 1:

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.
Using a floating pragma may introduce several vulnerabilities if compiled with an older version.
The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.

## Informative

## 1)

**PRESENCE OF OVERPOWERED ROLE**

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.
Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

### Remediation # 2:

We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address. For systems that are provisioned for a single user.

## 2)

**HARD-CODED ADDRESS DETECTED**

The contract contains an unknown hard-coded address. This address might be used for some malicious activity. Please check the hard-coded address and its usage.
These hard-coded addresses may be used everywhere throughout the code to define states and interact with the functions and external calls.
Therefore, it is extremely crucial to ensure the correctness of these token contracts as they define various important aspects of the protocol operation.
A misconfigured address mapping could lead to the potential loss of user funds or compromise of the contract owner depending on the function logic.

### Remediation # 2:

It is required to check the address. Also, it is required to check the code of the called contract for vulnerabilities.
Ensure that the contract validates if there's an address or a code change or test cases to validate if the address is correct.

# GAS

## 1)

**CHEAPER INEQUALITIES IN REQUIRE()**

The contract was found to be performing comparisons using inequalities inside the `require` statement. When inside the `require` statements, non-strict inequalities `(>=, <=)` are usually costlier than strict equalities `(>, <)`.

### Remediation # 1:

It is recommended to go through the code logic, and, if possible, modify the non-strict inequalities with the strict ones to save `~3` gas as long as the logic of the code is not affected.

## 2)

**FUNCTION SHOULD BE EXTERNAL**

A function with `public` visibility modifier was detected that is not called internally.
`public` and `external` differs in terms of gas usage. The former use more than the latter when used with large arrays of data. This is due to the fact that Solidity copies arguments to memory on a `public` function while `external` read from calldata which a cheaper than memory allocation.

### Remediation # 1:

If you know the function you create only allows for `external` calls, use the `external` visibility modifier instead of `public`. It provides performance benefits and you will save on gas.

## 3)

**LONG REQUIRE/REVERT STRINGS**

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

### Remediation # 1:

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

# Conclusion

**Smart contracts contain High severity issues! Liquiditypair contract's security is not checked due to out of scope.**

**Liquidity locking details NOT provided by the team.**

**Itish note:**

**Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.**