# Protocol Audit Report

Version 1.0

*[okeadev](https://github.com/OkeA-dev)*

October 15, 2024

# Santa's List Audit Report

Okea_dev.io

October 15, 2024

Prepared by: Okea_dev Lead Security Researcher: - Oke Abdulquadri O.

## Table of Contents

    * [H-5] Malicious code injection in Solmate ERC20 contract in `transferFrom` function which is inherited in `SantaToken`

  - Medium
    * [M-1] Cost to buy Nft via `SantaToken::buypresent` is 2e18 SantaToken but it burn only 1e18 amount of SantaToken.
  - Low
    * [L-1] `SantasList::collectPresent` is available after Christmas, allowing NICE or EXTRANICE users to collect presents anytime afterward.

## Protocol Summary

Santa's List is the main contract that stores the list of naughty and nice people. It doubles as an NFT contract that people can collect if they are NICE or EXTRA_NICE. In order for someone to be considered NICE or EXTRA_NICE they must be first "checked twice" by Santa.

Once they are checked twice, NICE users can collect their NFT, and EXTRA_NICE users can collect their NFT and they are given SantaTokens. The SantaToken is an ERC20 that can be used to buy the NFT for their NAUGHTY or UNKNOWN friends.

## Disclaimer

The OkeA_dev team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

## Risk Classification

| | | Impact | | |
|---|---|---|---|---|
| | | High | Medium | Low |
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |

| | Impact | | | |
|---|---|---|---|---|
| | Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

## Audit Details

### The findings described in this document correspond with this commit hash

```
7cf6dddb7ea37333c86a4bbda7cae8ce0ca01f9f
```

### Scope

```
./src/
#-- SantaToken.sol
#-- SantasList.sol
#-- TokenUri.sol
```

### Roles

- `Santa` - Deployer of the protocol, should only be able to do 2 things:
  - `checkList` - Check the list once
  - `checkTwice` - Check the list twice
  - Additionally, it's OK if Santa mints themselves tokens.
- `User` - Can buyPresents and mint NFTs depending on their status of NICE, NAUGHTY, EXTRA-NICE or UNKNOWN # Executive Summary The audit was conducted on Santa's List smart contracts, aimed at ensuring the security and functionality of the system. The objective was to identify vulnerabilities and ensure compliance with industry best practices for Nft,and token protocols. ## Issues found

| Severities | Numbers of issues found |
|---|---|
| High | 5 |

| Severities | Numbers of issues found |
|---|---|
| Medium | 1 |
| Low | 1 |
| Info | 0 |
| Total | 7 |

# Findings

## High

### [H-1] Incorrect default enum setting all addressess status to NICE, meaning anyone can collect present.

**Description:** An "Incorrect default enum" vulnerability occurs when the enum controlling the status of all addresses in the smart contract defaults to the first value, in this case this "NICE". This meaning all addresses are automatically granted "NICE" by default, without proper validations. As a result, any addresses can collect present only intended for those with "NICE" status. This flaw compromises the intended logic of the contract and allow unauthorized users to exploits the contract.

**Impact:** Unauthorized users can collect present without proper checklist by the Santas.

**Proof of Concept:** The following Foundry test will show that any user is able to call `SantasList::collectPresent` function after `CHRISTMAS_2023_BLOCK_TIME = 1_703_480_381`

```
1     function testCollectPresentWithoutGettingChecklist() public  {
2         //prank an attacker address
3         vm.startPrank(attacker);
4         vm.warp(santasList.CHRISTMAS_2023_BLOCK_TIME() + 1);
5         //collect present without any check from Santa
6         santasList.collectPresent();
7         assertEq(santasList.balanceOf(attacker), 1);
8         vm.stopPrank();
9     }
```

**Recommended Mitigation:** I suggest to modify `Status` enum, and use `NAUGHTY` status as first one. This way, all users will defaults to UNKNOWN status, preventing the successful call to `collectPresent` before any check of Santas.

```
1   enum Status {
2         NAUGHTY,
```

```
        NOT_CHECKED_TWICE
        NICE,
        EXTRA_NICE,
    }
```

Once `Status` is modify than the collect present can no longer be called successfully using the aforementioned Proof of Concept.

### [H-2] The `SantasList::checkList` lacks access control, allowing anyone to set the Status to `NICE` and collect presents.

**Description:** The `SantasList::checkList` function in the smart contract lack proper access control, enabling unauthorized user to change an address's status to NICE. Only Santa should have access to `SantasList::checkList` function, but anyone is allowed to update status and collect rewards or present without restrictions. **Impact:** Anyone can unauthorizedly update status to "NICE" and collect an NFT without Santas checklisting their address.

**Proof of Concept:** The following foundry test would show attacker can update an address's status and collect Nft without any restrictions.

```
    function testUnauthorizedStatusUpdateAndCollectReward() public {
        vm.startPrank(attacker);
        santasList.checkList(attacker, SantasList.Status.NICE);

        vm.warp(santasList.CHRISTMAS_2023_BLOCK_TIME() + 1);

        santasList.collectPresent();
        assertEq(santasList.balanceOf(attacker), 1);
        vm.stopPrank();

    }
```

**Recommended Mitigation:** Add access control measure to the `SantasList::checklist`, to restrict unauthorized access.

```
// Add onlysanta modifier to this function
@>   function checkList(address person, Status status) external
    onlySanta {
        s_theListCheckedOnce[person] = status;
        emit CheckedOnce(person, status);
    }
```

### [H-3] `SantasList::buyPresent` burns the presentReceiver's tokens and sends the presents to the caller instead of the presentReceiver.

**Description:** The `SantasList::buyPresent` function implements incorrect logic for buying presents. It is intended to burn the caller's tokens and mint an NFT for the presentReceiver. However, it mistakenly burns the presentReceiver's tokens and mints the NFT to the caller's address.

**Impact:** presentReciever unknowingly lose thier token existing, if they have one, and new NFT is minted to the caller address.

**Proof of Concept:** The following foundry test framework shows how the incorrect logic applies.

```
    function test_AttackerCanMintNft_ByBurningTokensOfOtherUsers()
        public {
    // address of the attacker
    address attacker = makeAddr("attacker");

    vm.startPrank(santa);
    // Santa checks user once as EXTRA_NICE
    santasList.checkList(user, SantasList.Status.EXTRA_NICE);
    santasList.checkTwice(user, SantasList.Status.EXTRA_NICE);
    vm.stopPrank();

    // christmas time  HO-HO-HO
    vm.warp(santasList.CHRISTMAS_2023_BLOCK_TIME());

    // User collects their NFT and tokens for being EXTRA_NICE
    vm.prank(user);
    santasList.collectPresent();

    assertEq(santaToken.balanceOf(user), 1e18);

    uint256 attackerInitNftBalance = santasList.balanceOf(attacker);

    // attacker get themselves the present by passing presentReceiver
        as user and burns user's SantaToken
    vm.prank(attacker);
    santasList.buyPresent(user);

    // user balance is decremented
    assertEq(santaToken.balanceOf(user), 0);
    assertEq(santasList.balanceOf(attacker), attackerInitNftBalance +
        1);
    }
```

**Recommended Mitigation:** 1. Burn the SantasToken of the caller, i.e msg.sender. 2. Mint NFT to presentReceiver

```
+ function _mintAndIncrementReceiver(address presentReceiver) private {
```

```
+         _safaMint(presentReceiver, s_tokenCounter++);
+     }

  function buyPresent(address presentReceiver) external {
-         i_santaToken.burn(presentReceiver);
-         _mintAndIncrement();
+         i_santaToken.burn(msg.sender);
+         _mintAndIncrementReceiver(presentReceiver);
  }
```

Implementing this recommendation would enable the `SantasList::buyPresent` function to execute the correct logic by burning the caller's santaToken and minting an NFT for the presentReceiver.

### [H-4] Multiple Calls to `SantasList::buyPresent` by NICE and EXTRANICE addresses, leading to unauthorized present collection.

**Description:** Addresses marked as NICE or EXTRANICE are able to repeatedly call the `SantasList::collectPresent` function after being checklisted once, allowing them to collect multiple presents without restriction. This results in unauthorized accumulation of presents, which undermines the intended single-collection rule for each address.

**Impact:** Unauthorized collection of multiple presents by addresses after being checklisted once.

**Proof of Concept:** The following foundry test shows how EXTRANICE user can collect present multiple with out restrictions.

```
    function testMultipleCollectPresentByNiceOrExtra() public {
        address attacker = makeAddr("attacker");

        vm.startPrank(santa);
        santasList.checkList(attacker, SantasList.Status.EXTRA_NICE);
        santasList.checkTwice(attacker, SantasList.Status.EXTRA_NICE);
        vm.stopPrank();

        vm.warp(santasList.CHRISTMAS_2023_BLOCK_TIME());

        uint256 attackerInitialNFTBalance = santasList.balanceOf(
            attacker);
        assertEq(attackerInitialNFTBalance, 0);

        vm.startPrank(attacker);
        santasList.collectPresent();
        santasList.safeTransferFrom(attacker, makeAddr("attacker2"), 0)
            ;
        santasList.collectPresent();

```

```
19          vm.stopPrank();
20
21          assertEq(santaToken.balanceOf(attacker), 2e18);
22      }
```

**Recommended Mitigation:** To prevent unauthorized repeated present collections after an address has been checklisted, mark the address as 'hasClaimed' once it collects its present. Additionally, check if 'hasClaimed' is true, and if so, revert with 'Santa__HasAlreadyClaimed'. This will prevent the address from collecting any more presents.

```
1  //  ERROR //
2  +   error SantasList__HasAlreadyClaimed();
3  //  STATE VARIABLE //
4  +   mapping(address => bool) private hasClaimed;
5
6      function collectPresent() external {
7  +           if (hasClaimed) {
8  +            revert SantasList__HasAlreadyClaimed();
9  +           }
10             if (block.timestamp < CHRISTMAS_2023_BLOCK_TIME) {
11                 revert SantasList__NotChristmasYet();
12             }
13             if (balanceOf(msg.sender) > 0) {
14                 revert SantasList__AlreadyCollected();
15             }
16             if (s_theListCheckedOnce[msg.sender] == Status.NICE &&
                   s_theListCheckedTwice[msg.sender] == Status.NICE) {
17                 _mintAndIncrement();
18 +               hasClaimed[msg.sender] = true;
19                 return;
20             } else if (
21                 s_theListCheckedOnce[msg.sender] == Status.EXTRA_NICE
22                     && s_theListCheckedTwice[msg.sender] == Status.
                           EXTRA_NICE
23             ) {
24                 _mintAndIncrement();
25                 i_santaToken.mint(msg.sender);
26 +               hasClaimed[msg.sender] = true;
27                 return;
28             }
29             revert SantasList__NotNice();
30     }
```

### [H-5] Malicious code injection in Solmate ERC20 contract in `transferFrom` function which is inherited in `SantaToken`

## Medium

### [M-1] Cost to buy Nft via `SantaToken::buypresent` is 2e18 SantaToken but it burn only 1e18 amount of SantaToken.

## Low

### [L-1] `SantasList::collectPresent` is available after Christmas, allowing NICE or EXTRANICE users to collect presents anytime afterward.

**Description:** The `SantasList::collectPresent` function becomes active after Christmas, enabling users with NICE or EXTRANICE status to collect their presents at any time after the holiday. This ensures flexibility for users to claim their rewards whenever they choose, without a strict time limit. Whereas, the function is expected to revert 24 hours after chrismas.

code

```
function collectPresent() external {
    @>  if (block.timestamp < CHRISTMAS_2023_BLOCK_TIME) {
            revert SantasList__NotChristmasYet();
        }
        if (balanceOf(msg.sender) > 0) {
            revert SantasList__AlreadyCollected();
        }
        if (s_theListCheckedOnce[msg.sender] == Status.NICE &&
           s_theListCheckedTwice[msg.sender] == Status.NICE) {
            _mintAndIncrement();
            return;
        } else if (
            s_theListCheckedOnce[msg.sender] == Status.EXTRA_NICE
                && s_theListCheckedTwice[msg.sender] == Status.
                   EXTRA_NICE
        ) {
            _mintAndIncrement();
            i_santaToken.mint(msg.sender);
            return;
        }
        revert SantasList__NotNice();
    }
```

**Impact:** NICE and EXTRANICE can collect present anytime after christmas.

**Proof of Concept:** The following foundry test show how to collect present after christmas as pasted.

```
function testCollectPresentAnytimeAfterChristmas() public {
     vm.startPrank(santa);
        santasList.checkList(user, SantasList.Status.NICE);
        santasList.checkTwice(user, SantasList.Status.NICE);
        vm.stopPrank();

        // add one whole week to christmas time.
        vm.warp(santasList.CHRISTMAS_2023_BLOCK_TIME() + 1 weeks );

        vm.startPrank(user);
        santasList.collectPresent();
        assertEq(santasList.balanceOf(user), 1);
        vm.stopPrank();
}
```

**Recommended Mitigation:** I suggest to modify `SantasList::collectPresent` for restrict collection before and after christmas day.

```
function collectPresent() external {
-       if (block.timestamp < CHRISTMAS_2023_BLOCK_TIME) {
+       if (block.timestamp < CHRISTMAS_2023_BLOCK_TIME && block.timestamp > CHRISTMAS_2023_BLOCK_TIME + 1 days) {
            revert SantasList__NotChristmasYet();
        }
        if (balanceOf(msg.sender) > 0) {
            revert SantasList__AlreadyCollected();
        }
        if (s_theListCheckedOnce[msg.sender] == Status.NICE && s_theListCheckedTwice[msg.sender] == Status.NICE) {
            _mintAndIncrement();
            return;
        } else if (
            s_theListCheckedOnce[msg.sender] == Status.EXTRA_NICE
                && s_theListCheckedTwice[msg.sender] == Status.EXTRA_NICE
        ) {
            _mintAndIncrement();
            i_santaToken.mint(msg.sender);
            return;
        }
        revert SantasList__NotNice();
    }
```