Santa’s List Initial Audit Report

Version 1.0

Alchemist

January 23, 2026

Santa’s List Audit Report

Aditya Kumar Mishra

Jan 23, 2026

Santa’s List Audit Report

Prepared by: Team Alchemist

Lead Auditors: - Aditya Kumar Mishra

Assisting Auditors: - None

Table of contents

About Alchemist

We are Team Alchemist, currently a one-member team founded by Aditya Kumar Mishra. Our focus is on learning, researching, and implementing smart contract security, with an emphasis on understanding vulnerabilities, best practices, and secure blockchain development.

Disclaimer

The Alchemist 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
	Low	M	M/L	L

Audit Details

The findings described in this document correspond to the following commit hash:

```
5c7e243
```

Scope

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

Protocol Summary

Santa's List is a protocol for tracking naughty and nice individuals on-chain. Santa maintains a dual-check system where addresses must be marked as NICE or EXTRA_NICE twice before becoming eligible to collect NFT presents after Christmas 2023. Extra nice individuals also receive SantaTokens.

Roles

- **Santa**: The protocol owner who maintains the naughty/nice list. Only Santa should mark addresses with status checks (denominated by `i_santa` variable).
- **Nice List Members**: Addresses checked twice as NICE by Santa, eligible to collect one NFT after Christmas.
- **Extra Nice List Members**: Addresses checked twice as EXTRA_NICE by Santa, eligible to collect one NFT and SantaTokens after Christmas.
- **Token Holders**: Users who own SantaTokens and can purchase presents for others.

Executive Summary

This audit identified critical access control vulnerabilities and logic flaws in the Santa's List protocol. The most severe findings allow any user to bypass Santa's authority and claim free presents. The protocol suffers from enum default value issues and incorrect token burning logic.

Issues found

Severity	Number of issues found
High	3
Medium	1
Low	1
Info	0
Total	5

Findings

High

[H-1] No access controls on `checkList` function

Description: The `checkList` function is documented as "Only callable by santa" but lacks the `onlySanta` modifier. This allows any external caller to arbitrarily set any address's first status check to NICE or EXTRA_NICE, completely bypassing Santa's authority.

```
function checkList(address person, Status status) external {
    s_theListCheckedOnce[person] = status;
    emit CheckedOnce(person, status);
}
```

Impact: High - Attackers can self-assign NICE/EXTRA_NICE status. After Santa's `checkTwice` confirmation, they claim free NFTs/tokens via `collectPresent()`. Protocol's core access control model is broken.

Proof of Concept:

```
function test_Anyone_can_call_CheckList() public {
    address anyone = makeAddr("anyone");
    vm.prank(anyone);
    santasList.checkList(user, SantasList.Status.NICE);
    assertEq(
        uint256(santasList.getNaughtyOrNiceOnce(user)),
        uint256(SantasList.Status.NICE)
    );
}
```

Recommended Mitigation: Add `onlySanta` modifier:

```
-function checkList(address person, Status status) external {
+function checkList(address person, Status status) external onlySanta {
    s_theListCheckedOnce[person] = status;
    emit CheckedOnce(person, status);
}
```

[H-2] By default user status is `NICE` (unchecked addresses can claim presents)

Description: The `Status` enum starts with `NICE` as index 0. Uninitialized `mapping(address => Status)` defaults to `Status(0)` = `NICE`. Thus, any address Santa never checked has both `s_theListCheckedOnce` and `s_theListCheckedTwice` set to `NICE`, making them eligible for `collectPresent()` after Christmas.

```
enum Status {
    NICE,
    EXTRA_NICE,
    NAUGHTY,
    NOT_CHECKED_TWICE
}
```

Impact: High - Free NFT minting for unlimited addresses. Mass exploitation possible. Santa's authority completely bypassed. Violates core protocol invariant.

Proof of Concept:

```
function test_UncheckedAddressCanClaimNFT() public {
    address uncheckedUser = makeAddr("freshUser");
    vm.warp(santasList.CHRISTMAS_2023_BLOCK_TIME() + 1);

    vm.startPrank(uncheckedUser);
    santasList.collectPresent();
    vm.stopPrank();

    assertEq(uint256(santasList.getNaughtyOrNiceOnce(uncheckedUser)),
        0);
    assertEq(uint256(santasList.getNaughtyOrNiceTwice(uncheckedUser)),
        0);
    assertEq(santasList.balanceOf(uncheckedUser), 1);
}
```

Recommended Mitigation:

```
-enum Status {
-    NICE,
-    EXTRA_NICE,
-    NAUGHTY,
-    NOT_CHECKED_TWICE
-}
+enum Status {
+    UNKNOWN,
+    NICE,
+    EXTRA_NICE,
+    NAUGHTY
+}
```

[H-3] `buyPresent` burns from receiver, mints to caller

Description: `buyPresent(presentReceiver)` burns 1 SantaToken from `presentReceiver` but mints NFT to `msg.sender`. This is backwards from documented intent ("anyone with SantaTokens" pays).

```
function buyPresent(address presentReceiver) external {
    i_santaToken.burn(presentReceiver);
    _mintAndIncrement();
}
```

Impact: High - Griefing attack. Arbitrary token burning from unsuspecting addresses. Economic inversion - receiver pays, caller receives NFT.

Proof of Concept:

```
function test_buyPresentBurnsFromReceiverNotCaller() public {
    address payer = makeAddr("payer");
    address receiver = makeAddr("receiver");

    deal(address(santaToken), receiver, 10 ether);

    vm.prank(payer);
    santasList.buyPresent(receiver);

    assertEq(santaToken.balanceOf(receiver), 9 ether);
    assertEq(santasList.balanceOf(payer), 1);
}
```

Recommended Mitigation:

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

Medium

[M-1] `buyPresent` hardcoded 1e18 burn + wrong payer

Description: `buyPresent` burns exactly 1e18 SantaTokens from `presentReceiver` (not caller) but mints NFT to `msg.sender`. No configurable cost or allowance check.

Impact: Medium - Griefing vector. Economic inversion. Magic number reduces flexibility.

Proof of Concept:

```
function test_buyPresentUsesHardcodedAmount() public {
    address payer = makeAddr("payer");
```

```
    address receiver = makeAddr("receiver");

    deal(address(santaToken), receiver, 10 ether);

    vm.prank(payer);
    santasList.buyPresent(receiver);

    assertEq(santaToken.balanceOf(receiver), 9 ether);
    assertEq(santasList.balanceOf(payer), 1);
}
```

Recommended Mitigation:

```
+uint256 public constant PRESENT_COST = 1e18;

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

Low

[L-1] SantaToken constructor missing zero-address validation

Description: The `SantaToken` constructor accepts an `address santasList` parameter without validating it is non-zero. If deployed with `address(0)`, all functions depending on `i_santasList` become unusable.

```
constructor(address santasList) {
    i_santasList = santasList;
}
```

Impact: Low - Contract becomes permanently unusable if deployed with zero address. Requires redeployment to fix. Controlled by deployer.

Proof of Concept:

```
function test_ZeroSantaListBreaksSantaToken() public {
    address ZERO = address(0);
    SantaToken badToken = new SantaToken(ZERO);

    vm.expectRevert();
    badToken.mint(address(this));
```

```
}
```

Recommended Mitigation:

```
constructor(address santasList) {
+    require(santasList != address(0), "SantasList cannot be zero");
    i_santasList = santasList;
}
```

Informational / Non-Critical

No informational findings.

Gas (Optional)

No gas optimizations identified. "'