SantasList Audit Report

Version 1.0

0xBlockPay

September 27, 2024

SantasList Audit Report

0xBlockPay

September 27, 2024

Prepared by: 0xBlockPay Lead Security Researcher: - eth0x

Table of Contents

Protocol Summary

Santa has adopted blockchain technology to keep track of the naughty and nice! Hooray! So Santa hired some south pole contract elves to help build him a blockchain-based list. He'd never worked with blockchain elves before but was excited to see what they could do!

Disclaimer

The 0xBlockPay 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
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

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

Audit Details

Scope

The findings describe in this document correspond the following repository and commit hash:

```
https://github.com/Cyfrin/2023-11-Santas-List
```

```
Commit Hash: 91c8f8c94a9ff2db91f0ab2b2742cf1739dd6374
```

In Scope:

```
./src/
|__ SantaToken.sol
|__ SantasList.sol
|__ TokenUri.sol
```

- Solc Version: 0.8.22
- Chain(s) to deploy contract to: Arbitrum

Roles

- Santa: Deployer of the protocol, should only be able to do 2 things `checkList` - Check the list once and `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

*We spend 10 hours with 1 auditor using foundry, aderyn and slither tools.

Issues found

Severity	Number of issues found
High	6
Medium	1
Low	4
Info	1
Gas	2
Total	14

Findings

High

SantasListTest.t.sol

[H-1] Function `testPwned` allows tests to execute arbitrary programs on your computer

This function is unnecessary as it falls outside the scope of the protocol tests. Enabling this cheat code poses significant security risks to your project, as it allows tests to execute arbitrary programs on your computer, such as ransomware or other malicious software.

Additionally, to run this part of the code, it is necessary to change `HEVM_ADDRESS` to `VM_ADDRESS`.

Recommended Mitigation:

Remove this part of code:

```
_CheatCodes cheatCodes = _CheatCodes(VM_ADDRESS);

function testPwned() public {
  string[] memory cmds = new string[](2);
  cmds[0] = "touch";
  cmds[1] = string.concat("youve-been-pwned");
  cheatCodes.ffi(cmds);
}
```

SantasList.sol

[H-2] Uninitialized State Variables `s_tokenCounter`

This variable is crucial within the scope of the `SantaToken`. It represents the `ID` of an NFT Token. Using the same value can trigger a revert function in the protocol, potentially leading to a DDoS attack.

Solidity does initialize variables by default when you declare them, however it's good practice to explicitly declare an initial value. For example, if you transfer money to an address we must make sure that the address has been initialized.

Recommended Mitigation:

Add variable initialziation to `SantasList's` constructor

```
constructor(address santasList) ERC20("SantaToken", "SANTA", DECIMALS)
    {
```

```
  i_santasList = santasList;
  s_tokenCounter = 0;
}
```

[H-3] `block.timestamp` as a variable to unlock protocol

```
uint256 public constant CHRISTMAS_2023_BLOCK_TIME = 1_703_480_381

if (block.timestamp < CHRISTMAS_2023_BLOCK_TIME) {
  revert SantasList__NotChristmasYet();
}
```

Using block.timestamp as a parameter for an unlock protocol can be risky because this parameter can be manipulated by a dishonest miner. It might be better to use an enable or lock/unlock pattern instead.

Recommended Mitigation:

```
boolean private s_lock;

constructor(boolean _s_lock) {
  s_lock = _s_lock;
}

function setUnlock(boolean _s_lock) onlySanta {
  s_lock = _s_lock;
}

function collectPresent() external {
  if (s_lock) {
    revert SantasList__NotChristmasYet();
  }
  ...
}
```

[H-4] The variable PURCHASED_PRESENT_COST is declared but not used.

This step is much deeper than it appears. Normally, the best approach would be to remove this variable, but in this case, it is very important. It represents the cost of the SantaToken for potential buyers who want to purchase presents for NAUGHTY users.

Recommended Mitigation:

Please check H-5

[H-5] Function buyPresent should be rethought, function is differnet than specnet.

It is necessary to add check statment for user's status, that friends are NAUGHTY or UNKNOWNa. The Burn function burns presentReciver's tokens insted of msg.sender, In mint function to argument is for msg.sender insted of presentReciverand PURCHASED_PRESENT_COST is not used, now value is hardcoded = 1e18. In addition s_tokenCounter is not initialized properly H-2.

Recommended Mitigation:

```
 error SantaToken__ReciverIsNICEorEXTRA_NICE();

function buyPresent(address presentReceiver) external {

        if(s_theListCheckedOnce[presentReceiver] == Status.NICE ||
           s_theListCheckedOnce[presentReceiver] == Status.EXTRA_NICE
        )
          revert SantaToken__ReciverIsNICEorEXTRA_NICE():
        }

        i_santaToken.burn(msg.sender, PURCHASED_PRESENT_COST);

        _mintAndIncrement();
    }
```

In SantaToken.sol should be modified burn function

```
function burn(address from, uint256 cost) external {
  if (msg.sender != i_santasList) {
    revert SantaToken__NotSantasList();
  }
  _burn(from, cost);
}
```

Please note that in ERC20.sol, the _burn function signature includes two parameters: _burn(address, uint256). Additionally, please review the _mintAndIncrement function. The function name suggests that it first mints the token and then increments its ID. However, in reality, the ID is incremented first, and then the token is minted. Using the same ID can cause the protocol to revert. Additionally, a DDoS attack on the protocol is possible. Please check [H-6] for more details.

[H-6] DDOS and Reentrancy attack _mintAndIncrement uses _safeMint(address,uint256) and wrong logic

A msg.sender can be a smart contract that has not implemented onERC721Received interface. This will cause the transaction to revert, potentially blocking the protocol through DDoS attacks. The function also mint new NFT token on `msg.sender` instead of reciwer.

Recommended Mitigation:

Here you can check that recipient address is not contract or has implemented `IERC721Receiver` interface.

```
function isContract(address account) internal view returns (bool) {
  uint256 size;
  assembly { size := extcodesize(account) }
  return size > 0;
}

function _incrementAndMint(address to) public {
  require(!isContract(to) || IERC721Receiver(to).onERC721Received.selector == IERC721Receiver(to).onERC721Received(address(0), address(0), 0, ""), "Recipient contract does not implement onERC721Received");
  _safeMint(to, s_tokenCounter++);
}
```

POC The Re-etrancy attack is possible by this code:

```
pragma solidity ^0.8.0;

interface ISantasList {
    function collectPresent() external;
}

contract ReentrancyAttack {
    ISantasList public santasList;
    bool public attackInProgress = false;

    constructor(address _santasListAddress) {
        santasList = ISantasList(_santasListAddress);
    }

    function attack() external {
        attackInProgress = true;
        santasList.collectPresent();
    }

    fallback() external payable {
        if (attackInProgress) {
            attackInProgress = false;
            santasList.collectPresent();
        }
    }
}
```

Recommended Mitigation:

To avoid such situation use PullPayment or ReetrancyGuard pattern ([PullPaymentAndReetrancyGuard](https://docs.openzeppelin.com/contracts/4.x/api/security))

Medium

[M-1] Lack of `onlySanta` modifier and change name of function to `addToCheckList`.

The function does not have the `onlySanta` modifier, allowing all users to use this function, add their own address, and set the status. The function's name can be misleading because it does not perform any checks; it only adds the user to the map.

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

Low

[L-1] Lack of UNKNOWN status in `Status` enum.

In the documentation, there is an `UNKNOWN status`, but it is not present in the enum.

Recommended Mitigation:

Add this status to `Status` enum

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

Please review the protocol architecture, as the `UNKNOWN` status can be no longer necessary. More information can be found in the GAS section.

[L-2] `tokenURI` Should be view because it reads `TOKEN_URI` from the contract and without arugment.

Instead of marking a function as public, consider marking it as external

```
function tokenURI() external view override returns (string memory) {
  return TOKEN_URI;
}
```

SantaToken.sol

[L-3] Constant value in `_mint` method

In `_mint` function should be constant variable insted of, `1e18` hardcoded value.

Similar situation is in `_burn` function, but here value should be as a argument of function `_burn(from, 1e18);`

Recommended Mitigation:

```
uint256 private constant MINT = 1e18;

function mint(address to) external {
  if (msg.sender != i_santasList) {
    revert SantaToken__NotSantasList();
  }

  _mint(to, MINT);
  }

  // @audit as a cost should be PURCHASED_PRESENT_COST
  // uint256 public constant PURCHASED_PRESENT_COST = 2e18;
  function burn(address from, uint256 cost) external {
  if (msg.sender != i_santasList) {
    revert SantaToken__NotSantasList();
  }
  _burn(from, cost);
    }
```

[L-4] Event is missing `indexed` fields

In Solidity, the indexed keyword is used in event definitions to make the parameters searchable. When you mark a parameter as indexed, it allows you to filter events by that parameter when querying logs. This is particularly useful for tracking specific events in a smart contract.

```
event CheckedOnce(address indexed person, Status status);
event CheckedTwice(address indexed person, Status status);
```

Informational

[I-1] Lack of `SantasList__SecondCheckDoesntMatchFirst.selector` in `vm.expectRevert()` in `testCantCheckListTwiceWithDifferentThanOnce` function.

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

Recommended Mitigation:

Add `SantasList__SecondCheckDoesntMatchFirst.selector` to the `vm.expectRevert()` function

```
  vm.expectRevert(SantasList__SecondCheckDoesntMatchFirst.selector);
```

Gas

[G-1] Unnecesary status `NOT_CHECKED_TWICE` in `Status` enum.

The status `NOT_CHECKED_TWICE` is not in documentation.

Recommended Mitigation:

Remove this status

[G-2] It is possible to remove the second list `s_theListCheckedTwice` and all checks connected with it.

Now the protocol is using two lists that operate on the same principle. It is possible to use only one list `s_theListCheckedOnce` to achieve the same functionality as currently. It will dramatically simplify the logic by removing the `checkTwice(address, Status)` function and reduce gas costs.

Recommended Mitigation:

```
mapping(address person => Status naughtyOrNice) private s_theListCheckedOnce;

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

function collectPresent() external {
  if (s_lock) {
    revert SantasList__NotChristmasYet();
  }

  if (balanceOf(msg.sender) > 0) {
    revert SantasList__AlreadyCollected();
  }

  if (s_theListCheckedOnce[msg.sender] == Status.NICE) {
    _incrementAndMint(msg.sender);
    return;
  } else if (
    s_theListCheckedOnce[msg.sender] == Status.EXTRA_NICE
    ) {
    _incrementAndMint(msg.sender);
    i_santaToken.mint(msg.sender);
    return;
  } else {
    revert SantasList__NotNice();
  }
}
```