Password Store Report

Performed by: *0xShiki*

Table of Contents

Protocol Summary

The “Baba Marta” protocol allows you to buy MartenitsaToken and to give it away to friends. Also, if you want, you can be a producer. The producer creates MartenitsaTokens and sells them. There is also a voting for the best MartenitsaToken. Only producers can participate with their own MartenitsaTokens. The other users can only vote. The winner wins 1 HealthToken. If you are not a producer and you want a HealthToken, you can receive one if you have 3 different MartenitsaTokens. More MartenitsaTokens more HealthTokens. The HealthToken is a ticket to a special event (producers are not able to participate). During this event each participant has producer role and can create and sell own MartenitsaTokens.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

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

Audit Details

The findings described in this document correspond to the following github repo:

```
https://github.com/Cyfrin/2024-04-Baba-Marta
```

Scope

```
- src
   - HealthToken.sol
   - MartenitsaEvent.sol
   - MartenitsaMarketplace.sol
   - MartenitsaToken.sol
   - MartenitsaVoting.sol
   - SpecialMartenitsaToken.sol
```

Roles

`Producer` - Should be able to create martenitsa and sell it. The producer can also buy martenitsa, make present and participate in vote. The martenitsa of producer can be candidate for the winner of voting.

`User` - Should be able to buy martenitsa and make a present to someone else. The user can collect martenitsa tokens and for every 3 different martenitsa tokens will receive 1 health token. The user is also able to participate in a special event and to vote for one of the producer's martenitsa.

Executive Summary

Issues found

Severity	Number of issues found
High	1
Medium	2
Low	1
Gas	0
Info	0
Total	4

Findings

High

H-01. Users can increment their token count with no restriction by calling `MartenitsaToken::updateCountMartenitsaTokensOwner` function, causing to increase their HealthToken balance

Relevant GitHub Links

https://github.com/Cyfrin/2024-04-Baba-Marta/blob/5eaab7b51774d1083b926bf5ef116732c5a35cfd/src/MartenitsaTo

Vulnerability Details

The purpose of the `HealthToken` token is to serve as a reward mechanism for participants who have more than 3 MartenitsaTokens. For every 3 different MartenitsaTokens they receive 1 HealthToken. Users can also join an event if they have sufficient amount of `HealthToken` tokens, where they can become producers during the event and be able to sell their `MartenitsaToken` NFTs. However, users can update `MartenitsaToken::countMartenitsaTokensOwner` mapping without restriction by calling `updateCountMartenitsaTokensOwner`, because it's marked as an `external` function. In that way, users can manipulate the contract by: 1. joining events 2. becoming producers 3. becoming sellers to sell their `MartenitsaToken` NFTs.

The other problem with this is that users can call `MartenitsaMarketplace::collectReward` and receive HealthTokens for every 3 different MartenitsaTokens they own, since `collectReward` relies on the `MartenitsaToken::countMartenitsaTokensOwner` mapping.

Impact

By incrementing their token ID count, users can falsely join events, become producers, and sell `MatenitsaToken` NFTs. Also, by incrementing the `MartenitsaToken::countMartenitsaTokensOwner` mapping, users can receive infinite HealthTokens by calling `MartenitsaMarketplace::collectReward`.

Tools Used

Manual Review

Proof of Code

Add this test to `MartenitsaMarketplace.t.sol`:

PoC

```
    function testUserCanIncreaseTheirTokenCount() public {
        // Bob update his token count to 3
        vm.startPrank(bob);
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        vm.stopPrank();
        console.log("Bob's Token Id count: ", martenitsaToken.
            getCountMartenitsaTokensOwner(bob));

        // Bob collect rewards for having a count of 3
        vm.startPrank(bob);
        marketplace.collectReward();
        vm.stopPrank();

        console.log("Bob's Health Token balance: ", healthToken.
            balanceOf(bob));

        // Bob update his token count to 6
        vm.startPrank(bob);
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        vm.stopPrank();
        console.log("-----------------------------");

        console.log("Bob's Token Id count: ", martenitsaToken.
            getCountMartenitsaTokensOwner(bob));

```

```
27          // Bob collect rewards for having a count of 6
28          vm.startPrank(bob);
29          marketplace.collectReward();
30          vm.stopPrank();
31
32          console.log("Bob's Health Token balance: ", healthToken.
                balanceOf(bob));
33      }
```

The logs from this test show that Bob increments his token ID count by 3 without restriction, and collects a reward of 1 `HealthToken`. Then, he increments his token ID count by 3 again, and collects another reward of 1 more `HealthToken`. resulting in owning a total of 2 HealthTokens.

Recommendations

Consider implementing a different logic in `MartenitsaToken` and `MartenitsaMarketplace` contracts. For instance, the `updateCountMartenitsaTokensOwner` function can be removed from the `MartenitsaToken` contract, and be implemented as an internal function in the `MartenitsaMarketplace` contract. In this way, the `MartenitsaMarketplace` contract can control the increment of the token ID count, and the reward mechanism can be implemented in a more secure way.

Medium

M-01. User can transfer `MartenitsaToken` NFT by calling `ERC721::transferFrom` function, causing the internal logic of user's martenitsa count to break

Vulnerability Details

The `MartenitsaToken` contract inherits the `ERC721` standard to manage the ownership of NFTs. By using the `ERC721::transferFrom` function, a user can transfer an NFT to another user. However, in this way, the user's martenitsa count is not updated correctly in the contract, leading to inconsistencies in the contract state.

Impact

As a result of transfering NFTs using the `transferFrom` function, the user's martenitsa count is not updated correctly, leading to inconsistencies in the contract state.

Tools Used

Manual Review

Proof of Code

Add this test to `MartenitsaToken.t.sol`:

PoC

```
    function testUserCanTransferToAnotherUserUsingTransferFrom() public {
        // Chasy lists a martenitsa for sale
        vm.startPrank(chasy);
        martenitsaToken.createMartenitsa("bracelet");
        marketplace.listMartenitsaForSale(0, 1 wei);
        vm.stopPrank();

        // Bob buys the martenitsa
        vm.prank(chasy);
        martenitsaToken.approve(address(marketplace), 0);
        vm.prank(bob);
        marketplace.buyMartenitsa{value: 1 wei}(0);

        assert(martenitsaToken.ownerOf(0) == bob);
        console.log("Bob's count of token Ids before transfer: ", martenitsaToken.getCountMartenitsaTokensOwner(bob));
        console.log("Jack's count of token Ids before transfer: ", martenitsaToken.getCountMartenitsaTokensOwner(jack));

        // Bob transfer his NFT to Jack
        vm.startPrank(bob);
        martenitsaToken.transferFrom(bob, jack, 0);
        vm.stopPrank();

        console.log("------------------------------");

        assert(martenitsaToken.ownerOf(0) == jack);
        console.log("Bob's count of token Ids after transfer: ", martenitsaToken.getCountMartenitsaTokensOwner(bob));
        console.log("Jack's count of token Ids after transfer: ", martenitsaToken.getCountMartenitsaTokensOwner(jack));
    }
```

From the logs of this test we can see that Bob can transfer his NFT to Jack. The ownership of the NFT is correct, but the `MartenitsaToken::getCountMartenitsaTokensOwner` mapping is not updated correctly.

Recommendations

Consider overriding the `transferFrom` function in the `MartenitsaToken` contract:

```
+    function transferFrom(address from, address to, uint256 tokenId)
     public override {
+        // Revert the transaction with an error message.
+        revert("KittyConnect: use safeTransferFrom for transfers");
+    }
```

M-02. `MartenitsaVoting::announceWinner` function miscalculates the winner of a voting event, causing the first one in the `MartenitsaVoting::tokenIds` array to always be the winner

Relevant GitHub Links

https://github.com/Cyfrin/2024-04-Baba-Marta/blob/5eaab7b51774d1083b926bf5ef116732c5a35cfd/src/MartenitsaVo

Vulnerability Details

The `MartenitsaVoting::announceWinner` function is used to calculate the winner of a voting event. The winner is the NFT with the most votes. However, if 2 or more NFTs have the same amount of votes, the first one in the `MartenitsaVoting::tokenIds` array will always be the winner. This is because the `MartenitsaVoting::announceWinner` function uses the `maxVotes` variable to store the maximum number of votes, and the `winner` variable to store the winner NFT ID. If the current NFT has more votes than the `maxVotes` variable, the `winner` variable is updated with the current NFT ID. However, if the current NFT has the same amount of votes as the `maxVotes` variable, the `winner` variable is not updated, and the first NFT in the `MartenitsaVoting::tokenIds` array is always the winner.

Impact

The winner of the voting event is always the first NFT in the `MartenitsaVoting::tokenIds` array if 2 or more NFTs have the same amount of votes, which leads to unfair results in voting events.

Tools Used

Manual Review

Proof of Code

Add this test to `MartenitsaVoting.t.sol`:

PoC

```
    function testWrongWinnerIfTwoHaveSameVotes() public {
        // Chasy lists a martenitsa for sale
        vm.startPrank(chasy);
        martenitsaToken.createMartenitsa("bracelet");
        marketplace.listMartenitsaForSale(0, 1 wei);
        vm.stopPrank();

        // Jack lists a martenitsa for sale
        vm.startPrank(jack);
        martenitsaToken.createMartenitsa("bracelet");
        marketplace.listMartenitsaForSale(1, 1 wei);
        vm.stopPrank();

        // Jack votes for Chasy's martenitsa
        vm.prank(jack);
        voting.voteForMartenitsa(0);

        // Bob votes for Jack's martenitsa
        vm.prank(bob);
        voting.voteForMartenitsa(1);

        // Announcing winner
        vm.warp(block.timestamp + 1 days + 1);
        vm.recordLogs();
        voting.announceWinner();

        console.log("NFT 0 votes >>> ", voting.getVoteCount(0));
        console.log("NFT 1 votes >>> ", voting.getVoteCount(1));
        Vm.Log[] memory entries = vm.getRecordedLogs();
        address winner = address(uint160(uint256(entries[0].topics[2]))
            );
        assert(winner == chasy);
    }
```

The logs from this test show how Jack votes for Chasy's martenitsa, and Bob votes for Jack's martenitsa. The winner of the voting event is Chasy's martenitsa, even though Jack's martenitsa has the same amount of votes.

Recommendations

Consider implementing a different logic for `MartenitsaVoting::announceWinner` function. For instance tokenIds, which have the same amount of votes can be stored in a newly created array.

Then the winning NFT can be selected from the newly create array, by implementing Chainlink VRF for determining the winner randomly.

Low

L-01. `MartenitsaVoting::announceWinner` function does not check if there are no votes, causing a participant to become a winner and receive a HealthToken

Relevant GitHub Links

https://github.com/Cyfrin/2024-04-Baba-Marta/blob/5eaab7b51774d1083b926bf5ef116732c5a35cfd/src/MartenitsaVo

Vulnerability Details

Users can vote for the best `MartenitsaToken` NFT, which is listed for sale on the `MartenitsaMarketplace`. The `MartenitsaVoting::announceWinner` function is used to calculate the winner of the voting event. The winner is the NFT with the most votes. However, the `MartenitsaVoting::announceWinner` function does not check if there are no votes, which a participant can receive a `HealthToken` even if there are no votes for the winner's NFT, and become the winner.

Impact

A participant can become a winner of the event and receive a `HealthToken` even if there are no votes for the winner's NFT, leading to unfair results in voting events.

Tools Used

Manual Review

Proof of Code

Add this test to `MartenitsaVoting.t.sol`:

PoC

```
function testIfThereAreNoVotes() public {
    // Chasy lists a martenitsa for sale
    vm.startPrank(chasy);
    martenitsaToken.createMartenitsa("bracelet");
    marketplace.listMartenitsaForSale(0, 1 wei);
    vm.stopPrank();

    // Jack lists a martenitsa for sale
    vm.startPrank(jack);
    martenitsaToken.createMartenitsa("bracelet");
    marketplace.listMartenitsaForSale(1, 1 wei);
    vm.stopPrank();

    // Announcing winner
    vm.warp(block.timestamp + 1 days + 1);
    vm.recordLogs();
    voting.announceWinner();

    console.log("Chasy's Health Token balance: ", healthToken.balanceOf
        (chasy));
    console.log("Jack's Health Token balance: ", healthToken.balanceOf(
        jack));

    Vm.Log[] memory entries = vm.getRecordedLogs();
    address winner = address(uint160(uint256(entries[0].topics[2])));
    assert(winner == chasy);
}
```

As we can see from the logs of this test the `HealthToken` balance of Chasy is 1. Jack's `HealthToken` balance is 0, because he is not the winner.

Recommendations

Consider adding a `require` statement to check if `maxVotes`, which is declared in `Martenitsa::announceWinner` function, is greater than 0:

```
    function announceWinner() external onlyOwner {
        require(block.timestamp >= startVoteTime + duration, "The
            voting is active");

        uint256 winnerTokenId;
        uint256 maxVotes = 0;

        for (uint256 i = 0; i < _tokenIds.length; i++) {
            if (voteCounts[_tokenIds[i]] > maxVotes) {
                maxVotes = voteCounts[_tokenIds[i]];
                winnerTokenId = _tokenIds[i];
            }
```

```
12          }
13
14  +       require(maxVotes > 0, "There are no votes");
15
16          list = _martenitsaMarketplace.getListing(winnerTokenId);
17          _healthToken.distributeHealthToken(list.seller, 1);
18
19          emit WinnerAnnounced(winnerTokenId, list.seller);
20      }
```