Baba Marta Audit Report

Version 1.0

Keyword

April 30, 2024

Baba Marta Audit Report

xKeywordx

April 30, 2024

Prepared by: xKeywordx

Table of Contents

Protocol Summary

Every year on 1st March people in Bulgaria celebrate a centuries-old tradition called the day of Baba Marta (“Baba” means Grandma and “Mart” means March), related to sending off the winter and welcoming the approaching spring. On that day and a few days afterwards, people exchange and wear the so-called “Martenitsa”. The martenitsa consists of decorative pieces of red and white twisted threads,

symbolising health and happiness. The martenitsas are given away to family and friends and are worn around the wrist or on clothes. The martenitsa is made of twined red and white threads – woolen, silk, or cotton. The most typical martenitsa represents two small dolls, known as Pizho and Penda. Pizho is the male doll, usually in white colour. Penda is the female doll, usually in red colour. Martenitsas come in a variety of shapes and sizes: bracelets, necklaces, tassels, pompoms and balls. The white is a symbol of purity, innocence, beauty and joy. The red is associated with health, vitality, fertility and bravery. According to the tradition, people wear martenitsas for a certain period, the end of which is usually associated with the first signs of spring – seeing a stork or a fruit tree in blossom.

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

I make all efforts to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit 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

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

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

Known issues

None

Issues found

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

Severity	Number of issues found

Findings

High

[H-1] Martenisa tokens count can be manipulated by anyone because the `MartenitsaToken::updateCountMartenitsaTokensOwner` function has no access control, and does not check the actual balance of the `owner` address before updating state.

Description: The `updateCountMartenitsaTokensOwner` function is supposed to keep track of the count of `martenitsaTokens` for a specific address. However, any user can call this function and pass in their own address or another user's address and update their balances.

Impact: A user can artificially inflate their balance to infinity, and mint an indefinite amount of `HealthToken` by calling the `MartenitsaMarketplace::collectReward` function or they can reduce the number of tokens of other users to 0.

Proof of Concepts: I updated the `MartenitsaToken::testUpdateCount`test inside `MartenisaToken.t.sol`. The test shows that `bob` can change `chasy's` balance however he wants.

Proof of Code

```
1       function testUpdateCount() public createMartenitsa {
2           vm.prank(chasy);
3           martenitsaToken.updateCountMartenitsaTokensOwner(chasy, "add");
4           assert(martenitsaToken.getCountMartenitsaTokensOwner(chasy) ==
                2);
5           vm.prank(bob);
6           martenitsaToken.updateCountMartenitsaTokensOwner(chasy, "add");
7           assert(martenitsaToken.getCountMartenitsaTokensOwner(chasy) ==
                3);
8           vm.prank(bob);
9           martenitsaToken.updateCountMartenitsaTokensOwner(chasy, "sub");
10          assert(martenitsaToken.getCountMartenitsaTokensOwner(chasy) ==
                2);
11      }
```

Recommended mitigation: Implement a balance check inside the `MartenitsaToken::updateCountMartenitsaTokensOwner` function in order to validate that the `owner` indeed has more or fewer tokens before updating the state. Maybe it is also worth considering adding access control.

[H-2] Wrong accounting inside `MartenitsaMarketplace` contract allows a user that holds 6 `martenitsaToken` to mint an indefinite amount of `HealthToken` via `MartenitsaMarketplace::collectReward` function.

Description: The `MartenitsaMarketplace::collectReward` function is supposed to allow users to collect 1 `HealthToken` for every 3 different `MartenitsaTokens` that they hold. If a user gets 6 tokens, they will be able to call `collectRewards` as many times as they want, and they will continue to receive `HealthTokens`.

Impact: A malicious user can mint an indefinite amount of `HealthTokens`.

Proof of Concepts: Paste this test inside `MartenitsaMarketplace.t.sol` file.

Proof of Code

```
    function testCollectReward() public eligibleForReward {
        vm.startPrank(bob);
        marketplace.collectReward();
        vm.stopPrank();
        uint256 balance = healthToken.balanceOf(bob);
        console2.log("Bob's balance is", balance);
        assert(balance == 10 ** 18);
        vm.startPrank(bob);
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        martenitsaToken.updateCountMartenitsaTokensOwner(bob, "add");
        marketplace.collectReward();
        uint256 balance2 = healthToken.balanceOf(bob);
        console2.log("Bob's balance is", balance2);
        marketplace.collectReward();
        uint256 balance3 = healthToken.balanceOf(bob);
        console2.log("Bob's balance is", balance3);
        marketplace.collectReward();
        uint256 balance4 = healthToken.balanceOf(bob);
        console2.log("Bob's balance is", balance4);
        marketplace.collectReward();
        uint256 balance5 = healthToken.balanceOf(bob);
        console2.log("Bob's balance is", balance5);
    }
```

Test output

```
Logs:
  Bob's balance is 1000000000000000000
  Bob's balance is 2000000000000000000
  Bob's balance is 3000000000000000000
  Bob's balance is 4000000000000000000
  Bob's balance is 5000000000000000000
```

Recommended mitigation: Increase the value of `MartenitsaMarketplace::_collectedRewards` mapping after each function call instead of setting it to be = `amountRewards`. Like this, the logic will also account for the past claims. Right now it is off by 1.

```
    function collectReward() external {
        //..
        //..
        uint256 amountRewards = (count / requiredMartenitsaTokens) -
            _collectedRewards[msg.sender];
        if (amountRewards > 0) {
-           _collectedRewards[msg.sender] = amountRewards;
+           _collectedRewards[msg.sender] += amountRewards;
            healthToken.distributeHealthToken(msg.sender, amountRewards
                );
        }
    }
```

[H-3] Buyers can donate money to the marketplace contract if they call the `MartenitsaMarketplace::buyMartenitsa` function with a `msg.value` amount that is higher than the `listing.price`. The contract doesn't refund excess funds sent.

Description: The `MartenitsaMarketplace::buyMartenitsa` function requires `msg.value >= listing.price`. If Chasy lists one item with a price of 1 Wei, and Bob wants to buy the item, but he accidentaly sends 5 Wei instead of 1, the transaction will not revert. Chasy will sell the item and she will receive 1 Wei, Bob will pay 5 Wei instead of 1, and the `MartenitsaMarketplace` contract will receive the extra 4.

Impact: Users can accidentally lose funds because the transaction will not revert if they call the `MartenitsaMarketplace::buyMartenitsa` with a `msg.value` amount that is > `listing.price`.

Proof of Concepts: Paste this test inside `MartenitsaMarketplace.t.sol` file.

Proof of Code

```
    function testBuyMartenitsaWrongAmount() public {
        vm.startPrank(chasy);
        martenitsaToken.createMartenitsa("bracelet");
        marketplace.listMartenitsaForSale(0, 1 wei);
        martenitsaToken.approve(address(marketplace), 0);
        vm.stopPrank();
        uint256 balance = bob.balance;
        console2.log("Bob's balance before sale is", balance);
        uint256 marketplaceBalanceBefore = address(marketplace).balance
            ;
```

```
        console2.log("Marketplace balance before sale is",
            marketplaceBalanceBefore);
        vm.prank(bob);
        marketplace.buyMartenitsa{value: 5 wei}(0);
        uint256 balance2 = bob.balance;
        console2.log("Bob's balance after sale is", balance2);
        uint256 marketplaceBalanceAfter = address(marketplace).balance;
        console2.log("Marketplace balance after sale is",
            marketplaceBalanceAfter);

        assert(martenitsaToken.ownerOf(0) == bob);
        assert(martenitsaToken.getCountMartenitsaTokensOwner(bob) == 1)
            ;
        assert(martenitsaToken.getCountMartenitsaTokensOwner(chasy) ==
            0);
    }
```

Test output

```
Logs:
  Bob's balance before sale is  5000000000000000000
  Marketplace balance before sale is  0
  Bob's balance after sale is  4999999999999999995
  Marketplace balance after sale is  4
```

Recommended mitigation: To prevent this from happening, the `require` statement inside the `MartenitsaMarketplace::buyMartenitsa` function should be changed to strict equality.

```
    function buyMartenitsa(uint256 tokenId) external payable {
        Listing memory listing = tokenIdToListing[tokenId];
        require(listing.forSale, "Token is not listed for sale");
-       require(msg.value >= listing.price, "Insufficient funds");
+       require(msg.value == listing.price, "Insufficient funds");
        //..
        //..
        //..
    }
```

Lows

[L-1] Failed buy transactions because the seller can transfer the token out before someone else buys it.

Description: The `MartenitsaMarketplace::listMartenitsaForSale` function doesn't transfer the listed tokens from the `seller` to the contract. Because of this, a `seller` can list a token for sale, and then transfer it to a different address. If they don't cancel the listing, the `buyer` will think that this is a legitimate listing. When they will try to buy this particular token, they will

get a failed transaction because the `safeTransferFrom` inside `MartenitsaMarketplace::buyMartenitsa` will fail.

Impact: Buyers will incur gas fees losses and keep getting failed transactions for what they deem as being a valid listing.

Proof of Concepts: Paste this test inside `MartenitsaMarketplace.t.sol` file and run it with the `forge test --mt testBuyMartenitsaAfterTransfer -vvv` command in order to see the logs. You will see that Chasy is able to transfer the token to Jack after listing it, and when Bob calls the `buyMartenitsa` function, the function will revert once it reaches the `safeTransferFrom` line of code because Chasy is not holding the token anymore.

Proof of Code

```
    function testBuyMartenitsaAfterTransfer() public {
        vm.startPrank(chasy);
        martenitsaToken.createMartenitsa("bracelet");
        marketplace.listMartenitsaForSale(0, 1 wei);
        martenitsaToken.approve(address(marketplace), 0);
        uint256 chasyBalanceBefore = martenitsaToken.balanceOf(chasy);
        console2.log("Chasy's balance before is", chasyBalanceBefore);
        martenitsaToken.transferFrom(chasy, jack, 0);
        uint256 jackBalance = martenitsaToken.balanceOf(jack);
        console2.log("Jack's balance is", jackBalance);
        uint256 chasyBalanceAfter = martenitsaToken.balanceOf(chasy);
        console2.log("Chasy's balance after is", chasyBalanceAfter);
        vm.stopPrank();
        vm.prank(bob);
        marketplace.buyMartenitsa{value: 1 wei}(0);
    }
```

Test output

```
Logs:
//..
//..
//..
@> ├─ [67871] MartenitsaMarketplace::buyMartenitsa{value: 1}(0)│├─
         [23585] MartenitsaToken::updateCountMartenitsaTokensOwner(bob:
            [0x1D96F2f6BeF1202E4Ce1Ff6Dad0c2CB002861d3e], "add")│└─←
             ()│├─
         [2143] MartenitsaToken::updateCountMartenitsaTokensOwner(chasy:
             [0x4f19A0AcB98b7aA30b0138D26e5E5F9634F32F5A], "sub")│└─←
             ()│├─
         emit MartenitsaSold(tokenId: 0, buyer: bob: [0
            x1D96F2f6BeF1202E4Ce1Ff6Dad0c2CB002861d3e], price: 1)│├─
         [0] chasy::fallback{value: 1}()│└─←
             ()
```

```
13 @> ├─     [3735] MartenitsaToken::safeTransferFrom(chasy: [0
      x4f19A0AcB98b7aA30b0138D26e5E5F9634F32F5A], bob: [0
      x1D96F2f6BeF1202E4Ce1Ff6Dad0c2CB002861d3e], 0)
```

Recommended mitigation: When listing the token this should be transferred from the seller to the Marketplace contract in order to prevent sellers from transferring them out before the actual sale or before they cancel their listing.

Info

[I-1] Sybil attack in `MartenitsaVoting::voteForMartenitsa` function. A bad actor can manipulate the outcome of the voting competition

Description: There are no checks/ access controls or costs inside the `MartenitsaVoting::voteForMartenitsa` function that will prevent a malicious user from creating an indefinite amount of wallets, and vote his own listing from all of them. The only things that this function checks are if `msg.sender` already voted, if the voting period has started and if the Martenitsa token is listed for sale.

Impact: A malicious user can create an indefinite number of new wallets and vote his own token in order to win the competition.

Recommended mitigation: Add additional enforcements in place such as, only martenitsa token holders can vote or only health token holders can vote, etc.

Gas

[G-1] Cache array length in order to save gas `MartenitsaVoting::announceWinner`, `MartenitsaEvent::stopEvent`

Recommended mitigation: Below you can find the instances.

```
    function announceWinner() external onlyOwner {
        //..
        //..
+       uint256 tokenIdsLength = _tokenIds.length;
-       for (uint256 i = 0; i < _tokenIds.length; i++) {
+       for (uint256 i = 0; i < tokenIdsLength; i++) {
            if (voteCounts[_tokenIds[i]] > maxVotes) {
                maxVotes = voteCounts[_tokenIds[i]];
                winnerTokenId = _tokenIds[i];
            }
        }
```

```
12          //..
13          //..
14      }
```

```
1       function stopEvent() external onlyOwner {
2           require(block.timestamp >= eventEndTime, "Event is not ended");
3   +       uint256 participantsLength = participants.length;
4   -       for (uint256 i = 0; i < participants.length; i++) {
5   +       for (uint256 i = 0; i < participantsLength; i++) {
6               isProducer[participants[i]] = false;
7           }
8       }
```