Oneshot Rapper Audit Report

Version 1.0

Zkillua

OneShot Rapper Audit Report

Zkillua

Mar 1st, 2024

Prepared by: [Zkillua]

Table of Contents

Table of Contents

Protocol Summary

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. Users mints a rapper NFT that begins with all the flaws and self-doubt we all experience. The only way to improve these stats is by staking. Experience on the streets will earn you Cred and remove your rapper's doubts. Users can put their Cred on the line to step on stage and battle their Rappers. Each rapper's skill is then used to weight their likelihood of randomly winning the battle!

Disclaimer

The Zkillua 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

Scope

- Code Base: https://github.com/Cyfrin/2024-02-one-shot
- In Scope:

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

Issues found

Severity	Number of issues found
High	2
Medium	1
Low	1

Severity	Number of issues found
Info	0
Gas Optimizations	0
Total	4

Findings

High

[H-1] Challenger Can participate in Battle without owning any RapperNft or cred tokens.

Description: - The RapBattle contract allows a challenger to participate in a battle without verifying that they own the NFT or have the required Cred tokens. This can be exploited by a challenger who initiates a battle with a random tokenId and matching credBet, without actually owning the NFT or Cred tokens. If the challenger wins, they receive the defender's Cred tokens. If they lose, the contract's attempt to transfer the challenger's Cred tokens to the defender fails, and the transaction reverts, leaving the defender uncompensated.

Impact:

- This vulnerability allows a challenger to gamble with no risk, potentially stealing the defender's Cred tokens if they win, and facing no loss if they lose, as the transaction will revert due to the failed token transfer. This defeats the whole purpose of RapperNft and staking.

Proof of Concept:

Add the following function in `OneShotTest.t.sol`

```
1
2  function test_AttackerCanBattleWithoutRapperNft() public
       twoSkilledRappers {
3
4          vm.startPrank(user);
5          oneShot.approve(address(rapBattle), 0);
6          cred.approve(address(rapBattle),4);
7          rapBattle.goOnStageOrBattle(0, 4); //tokenid=0,credbet=4
8          vm.stopPrank();
9
10         address attacker = makeAddr("attacker");
11         vm.startPrank(attacker);
```

```
12          console2.log("cred balance challenger ", cred.balanceOf(
                attacker));
13
14          rapBattle.goOnStageOrBattle(0, 4);// credBet=4 which attacker
                does not own.
15          console2.log("cred balance challenger ", cred.balanceOf(
                attacker));
16          assertEq(cred.balanceOf(attacker),4) // holds true whenever
                attacker wins the battle. transaction reverts otherwise.
17
18
19          vm.stopPrank();
20
21
22      }
```

- Defender calls goOnStageOrBattle with their NFT (tokenId) and Cred tokens (credBet), which are transferred to the contract.
- Challenger calls goOnStageOrBattle with any existing tokenId and the same credBet amount, without owning the NFT or Cred tokens.
- If the challenger wins, the credToken.transfer call succeeds, and they receive the defender's Cred tokens.
- If the challenger loses, the credToken.transferFrom call fails, the transaction reverts, and the defender receives nothing.

Recommended Mitigation:

- Modify the goOnStageOrBattle function to include checks that verify the challenger's ownership of the NFT and their possession of the required Cred tokens, as well as approval for the contract to spend those tokens. Implement the following checks for the challenger before initiating a battle: Check that the challenger owns the NFT and Check that the challenger has enough Cred tokens and has approved them for the contract.

```
1
2  ++ require(oneShotNft.ownerOf(_tokenId) == msg.sender, "Challenger must
       own the token");
3
4  ++ require(credToken.allowance(msg.sender, address(this)) >= _credBet,
       "Challenger must approve bet amount");
5  ++ require(credToken.balanceOf(msg.sender) >= _credBet, "Challenger
       must have enough Cred tokens");
```

[H-2] Predictable Random Number Generation (Miner Manipulability + Lack of Fairness)

Description: - The RapBattle contract uses a pseudo-random number generation technique that relies on block.timestamp, block.prevrandao, and msg.sender to determine the outcome of a battle. This method is considered weak and can be exploited by miners or validators, as they have some control over the block.timestamp and can potentially manipulate transaction ordering to achieve a desired outcome.

Impact: - The predictability and manipulability of the random number generation can lead to unfair battles, where the outcome may be influenced by miners or validators rather than being truly random. This compromises the integrity of the game and can result in a loss of trust from the players.

Proof of Concept: - A miner or validator, upon seeing a profitable outcome, could manipulate the block.timestamp within a certain range and choose whether or not to include a transaction in a block based on whether it results in a favorable outcome. This could be done by simulating the outcome off-chain before deciding on the actual block content.

Recommended Mitigation: - Replace the current pseudo-random number generation with a decentralized random number generator (RNG) such as Chainlink VRF (Verifiable Random Function). Chainlink VRF provides cryptographic proof of the randomness that can be verified on-chain, ensuring that the outcome is fair and cannot be tampered with by any party.

Medium

[M-1] BattlesWon by a user is not tracked (No State Update + Inaccurate Record Keeping)

Description: - The RapBattle contract does not maintain a record of the number of battles each user has won. Although there is an event Battle that indicates the winner of each battle, there is no state variable or mapping in the contract that keeps a tally of the victories for each participant. This results in the inability to retrieve or display the number of battles won by a user directly from the contract.

Impact: - Without accurate tracking of battles won, the contract lacks a crucial feature that could be used for leaderboards, rewards, or other game mechanics that depend on the number of victories. Users have no on-chain recognition of their achievements, and the game's competitive aspect is diminished.

Recommended Mitigation: - Integrate a mechanism to update the battlesWon attribute in the NFT metadata for the winning rapper's NFT after each battle.

Low

[L-1] Incorrect emit of Battle event (Boundary Condition Oversight + Incorrect Event Information)

Description: - The RapBattle contract emits a Battle event that may incorrectly identify the winner of a battle when the random number generated is exactly equal to the defenderRapperSkill. The conditional logic in the _battle function uses a less than or equal to (<=) comparison to determine if the defender wins. However, the event emission uses a strict less than (<) comparison, leading to a discrepancy where the defender should win (according to the logic), but the event indicates the challenger as the winner.

Impact: - This inconsistency can lead to confusion and disputes among players, as the event log will not accurately reflect the outcome determined by the contract's logic. This could undermine trust in the contract and negatively affect the user experience.

Proof of Concept:

- A battle occurs where random is exactly equal to defenderRapperSkill.
- The _battle function logic deems the defender the winner (random <= defenderRapperSkill).
- The Battle event is emitted with the challenger as the winner (random < defenderRapperSkill).

Recommended Mitigation: - Align the conditional check in the event emission with the logic used to determine the winner. Use the same <= comparison for both the logic and the event emission to ensure consistency.

Here's the corrected event emission logic within the _battle function:

```
1
2 // Emit the Battle event with the correct winner
3 ++ emit Battle(msg.sender, _tokenId, random <= defenderRapperSkill ?
      _defender : msg.sender);
4 // By making this change, the contract will emit the Battle event with
      the correct winner, matching the outcome as determined by the
      contract's logic, and eliminating any ambiguity in the battle
      results.
```