shieldify

Onchain Heroes

Season 2 (Dragma)

SECURITY REVIEW

Date: 4 August 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Onchain Heroes - Dragma

Onchain Heroes is a fully on-chain, idle RPG with novel on-chain game mechanics.

Introduction to Season 2

The next chapter in the ongoing battle for the multiverse. Picking up where Season 1 left off, Season 2 intensifies the fight as our heroes face an unprecedented threat: Dragma, the World Boss. With new mechanics, deeper strategy, and fresh rewards, this season introduces players to an evolved and expanded gameplay experience.

Season 2 Overview

- Progression Reset: All heroes have been reset to Level 1, providing a fresh start for all players.
- Hero Staking: Deploy your heroes into battle to earn $HERO tokens by defeating enemies and progressing through the season.
- New Damage System: Damage is now determined by a combination of hero level and weapon rarity, directly impacting player rewards.
- Weapon Traits: Weapons now feature Sharpness and Durability, which influence their effective damage output and how often they can be used in battle.
- Two-Phase Gameplay:
  - Phase 1: Available at launch, players will face Dragma's Underlings, preparing for the greater challenge ahead.
  - Phase 2: Unlocked several weeks after launch, this phase will bring the direct confrontation with Dragma and higher-stakes gameplay.
- Season Duration: Season 2 will run for approximately 8 to 12 weeks, offering ongoing challenges and evolving gameplay.

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4. Risk Classification

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 4 days, with a total of 64 hours dedicated by 2 researchers from the Shieldify team.

Overall, the code is well-written. The audit report flagged two High, one Medium and one Low severity findings concerning blocked gacha rewards, missing weapon shard rewards, failed dead-hero recovery, and unvalidated deadline parameters.

The Onchain Heroes team has been highly responsive to the Shieldify research team's inquiries and promptly implemented the recommendations.

5.1 Protocol Summary

Project Name	**Onchain Heroes - Dragma**
Repository	och-dragma
Type of Project	GameFi
Audit Timeline	4 days
Review Commit Hash	cb0645c3e7ca2a5aa7a6b9ce0263a182fba24fd9
Fixes Review Commit Hash	88dd42b03b5fa041e3b84a0bb93248041c538580

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/Dragma/Dragma.sol	531
src/Dragma/OCHEquipmentArmory.sol	194
Total	**725**

6. Findings Summary

The following issues have been identified, sorted by their severity:

- **High** issues: **2**
- **Medium** issues: **1**
- **Low** issues: **1**

ID	Title	Severity	Status
[H-01]	Hero Owner Can Never Receive Bronze Tier Gacha Tokens	High	Fixed
[H-02]	Hero Owner With Mythical Weapon Can Never Receive Weapon Shard Rewards	High	Fixed
[M-01]	Contract Owner Cannot Recover Dead Heroes Due to Conflicting State Validation Logic	Medium	Fixed
[L-01]	Deadline/End Params Not Validated Against `block.timestamp` or Minimum Value	Low	Acknowledged

7. Findings

[H-01] Hero Owner Can Never Receive Bronze Tier Gacha Tokens

Severity

High Risk

Description

The vulnerability exists in the `_calculateBonusRewards()` function within the gacha token distribution mechanism. The contract uses a packed data structure `BONUS_REWARD_TABLE` to store reward probabilities for different weapon rarities and token tiers. According to the commented specification, the data structure should contain probability values for four token tiers: `Rainbow` (bits 0-7), `Gold` (bits 8-15), `Silver` (bits 16-23), and `Bronze` (bits 24-31). However, the token distribution loop in the `_calculateBonusRewards()` function only iterates through the first three positions (i=0, i=1, i=2), accessing bits 0-7, 8-15, and 16-23, respectively.

The token ID assignment follows the formula gachaTokenId = (4 - i), which means:

- i=0 accesses bits 0-7 and assigns token ID 4
- i=1 accesses bits 8-15 and assigns token ID 3
- i=2 accesses bits 16-23 and assigns token ID 2

This creates a critical mismatch where the Bronze token data stored in bits 24-31 is never accessed, and consequently, Bronze tokens (presumably the lowest tier) are never distributed to players. The loop terminates after checking the first three tiers and uses a break statement upon finding a successful reward, ensuring that only one token type can be awarded per unstaking operation.

Location of Affected Code

File: src/Dragma/Dragma.sol#L569

```
function _calculateBonusRewards(uint256 heroId, uint256 requestId,
   uint256 randomNumber_) internal returns (uint256, uint256) {
    DragmaStorage storage $ = _getStorage();

    HeroInformation storage hero = $.heroInfo[heroId];

    // Get weapon id
    (, uint96 weaponId) = heroArmory.getArmory(heroId);

    // Get weapon data
    uint256 data = weaponData.getWeaponData(weaponId);

    // Extract `weaponData` from `data`.
    uint256 rarity = data & 0xff;

    uint256 zone = hero.attackZone;

    // gacha token rewards
    // accuracy upto 2dp
    uint256 rewards = (BONUS_REWARD_TABLE >> (rarity * 32)) & 0xffffffff;

    uint256 gachaTokenId;
    if (rewards != 0) {
        randomNumber_ = uint256(keccak256(abi.encode(randomNumber_,
           requestId)));
        // @audit bronze token doesnt get distributed
        for (uint256 i = 0; i < 3; i++) {
            uint256 chance = (((rewards >> (i * 8)) & 0xff) * (25 + (zone
                * 25))) / 10;
            if ((randomNumber_ % 10000) < chance) {
                // drop gacha token
                gachaTokenId = (4 - i);
                items.mint(hero.owner, gachaTokenId, 1);
                break;
            }
        }
    }

    // code
}
```

Impact

This vulnerability results in a systematic exclusion of Bronze-tier gacha tokens from the reward distribution system. Players who stake heroes and successfully complete unstaking operations will never receive Bronze tokens, regardless of their weapon rarity or zone selection.

Recommendation

The primary fix requires extending the reward distribution loop to include all four token tiers. Modify the loop condition in the `_calculateBonusRewards()` function from i < 3 to i < 4. This change will ensure that the Bronze token data stored in bits 24-31 of the `BONUS_REWARD_TABLE` is properly accessed and processed.

Team Response

Fixed.

[H-02] Hero Owner With Mythical Weapon Can Never Receive Weapon Shard Rewards

Severity

High Risk

Description

The vulnerability exists in the weapon shard reward calculation within the `_calculateBonusRewards()` function. The contract uses a packed constant `WEAPON_SHARD_TABLE (0xD282321E140A)` to store weapon shard probability values for different weapon rarities. This constant contains data for six rarity levels (Common through Legendary, rarities 0-5), with each rarity occupying 8 bits of data.

However, the weapon system supports seven rarity levels, including `Mythical weapons (rarity 6)`. When calculating weapon shard rewards for Mythical weapons, the function performs a bit shift operation `(WEAPON_SHARD_TABLE >> (6 * 8))` which shifts the constant right by 48 bits. Since the `WEAPON_SHARD_TABLE` constant only contains 48 bits of meaningful data, this shift operation results in accessing undefined data beyond the constant's scope, effectively returning zero.

The subsequent calculation `weaponShardChance = (0 * (25 + (zone * 25))) / 10` always evaluates to zero for Mythical weapons, causing the `guaranteedMint` to be zero and the conditional random chance check `if (weaponShardChance != 0)` to be skipped entirely. This results in Mythical weapon holders receiving zero weapon shards regardless of their attack zone or battle outcome.

Location of Affected Code

File: src/Dragma/Dragma.sol#L608

```
function _calculateBonusRewards(uint256 heroId, uint256 requestId,
    uint256 randomNumber_) internal returns (uint256, uint256) {
  // code
  uint256 weaponShardChance = (((WEAPON_SHARD_TABLE >> (rarity * 8)) & 0
     xff) * (25 + (zone * 25))) / 10;

  uint256 guaranteedMint = weaponShardChance / 1000;

  if (weaponShardChance != 0) {
      uint256 chance = weaponShardChance - (guaranteedMint * 1000);
      randomNumber_ = uint256(keccak256(abi.encode(randomNumber_,
         requestId)));
      if ((randomNumber_ % 1000) < chance) {
          guaranteedMint += 1;
      }
  }
  if (guaranteedMint > 0) {
      items.mint(hero.owner, 1, guaranteedMint);
  }
  return (gachaTokenId, guaranteedMint);
}
```

Impact

This vulnerability creates a significant imbalance in the reward distribution system, particularly affecting players who have invested in the highest-tier weapons. Mythical weapons, being the rarest and presumably most valuable weapons in the game, should logically provide the best or at least competitive weapon shard rewards. Instead, holders of these premium weapons receive no weapon shard rewards at all, creating a counterintuitive penalty for owning superior equipment.

Recommendation

The immediate fix requires updating the `WEAPON_SHARD_TABLE` constant to include weapon shard probability data for Mythical weapons (rarity 6). The current constant should be expanded from 48 bits to 56 bits to accommodate the seventh rarity level.

Team Response

Fixed.

[M-01] Contract Owner Cannot Recover Dead Heroes Due to Conflicting State Validation Logic

Severity

Medium Risk

Description

The `Dragma` contract contains a critical logic error in the `unstakeDiedHeroes()` function that creates an impossible execution condition, preventing the contract owner from recovering dead heroes. The vulnerability arises from conflicting state validation checks that fail to account for the hero state transitions that occur during the unstaking flow. The issue manifests through the following sequence of events. When users initiate hero unstaking through `unstakeMany()`, the contract sets the hero's stakeAt value to `MAX_UINT40_VALUE` to indicate a pending unstake request.

Subsequently, when the VRF callback processes the unstake request in `vrfCallback()`, heroes that die retain the `MAX_UINT40_VALUE` in their `stakeAt` field while having their `isDead` flag set to 1. The fundamental flaw occurs in the `unstakeDiedHeroes()` function, which includes a validation check that explicitly reverts when `hero.stakeAt == MAX_UINT40_VALUE.`

This creates an impossible logical condition where dead heroes that went through the normal unstaking process will always have `stakeAt` set to `MAX_UINT40_VALUE`, causing the recovery function to always revert with `AlreadyRequested()`. The function was designed to recover dead heroes, but fails to account for the state these heroes will be in after going through the standard unstaking flow.

Location of Affected Code

File: src/Dragma/Dragma.sol#L360

```
function unstakeMany(uint256[] calldata heroIds) external whenNotPaused {
    // check ownership
    uint256 len = heroIds.length;

    DragmaStorage storage $ = _getStorage();
    for (uint256 i = 0; i < len; i++) {
        uint256 id = heroIds[i];
        HeroInformation memory hero = $.heroInfo[id];

        if ((hero.owner != msg.sender)) revert CallerIsNotOwner();

        if (hero.stakeAt == MAX_UINT40_VALUE) revert AlreadyRequested();

        if ((hero.stakeAt + $.minimumBattleTime) > block.timestamp)
           revert BattleNotCompleted();

        if (hero.isDead != 0) revert HeroIsDead();

        // vrf request
        uint256 requestId = vrf.requestRandomNumberWithTraceId(0);

        $.heroInfo[id].stakeAt = uint40(MAX_UINT40_VALUE);
        $.request[requestId] =
            RequestData({owner: msg.sender, heroId: uint16(id),
               attackZone: hero.attackZone, fulfilled: 0});

        // emit unstake requested
        emit UnstakeRequested(msg.sender, id, requestId);
    }
}
```

Impact

The vulnerability completely disables the administrative recovery mechanism for dead heroes.

Recommendation

The vulnerability requires immediate correction of the state validation logic in the `unstakeDiedHeroes()` function to properly handle the expected state of dead heroes after they have gone through the unstaking request process. The primary fix involves removing or modifying the conflicting validation check that prevents the function from executing its intended purpose.

Team Response

Fixed.

[L-01] Deadline/End Params Not Validated Against `block.timestamp` or Minimum Value

Severity

Low Risk

Description

The `setSeasonTime()` function in the Dragma contract lacks input validation for the `startTime` and `endTime` parameters. Unlike other contracts in the OCH ecosystem (such as `Endgame.sol` and `LevelingGame.sol`), this function allows the owner to set timestamps in the past or create invalid time windows where `endTime < startTime`. This can lead to immediate disruption of game operations and create inconsistent season states that prevent normal protocol functionality.

Location of Affected Code

File: src/Dragma/Dragma.sol#L824

```
function setSeasonTime(uint40 startTime, uint40 endTime) external
   onlyOwner {
    DragmaStorage storage $ = _getStorage();
    $.startTime = startTime;  // No validation - could be in the past
    $.endTime = endTime;      // No validation - could be before
        startTime
}
```

Impact

This validation gap creates operational risks for the `Dragma` protocol's season management system. Setting invalid timestamp combinations can instantly disrupt core game functionality, leaving players unable to participate in expected activities. When administrators accidentally set an `endTime` in the past, the season appears immediately concluded, even if intended to remain active, blocking all staking operations and zone access. Similarly, configuring `endTime` before `startTime` creates an impossible temporal window where the season logic never recognizes an active state, effectively freezing protocol functionality. The impact falls through to other game functions like `_checkZoneIsStarted()`, `stakeMany()`.

Unlike other OCH contracts that implement proper timestamp validation, Dragma's lack of input checking leaves the protocol vulnerable to administrative errors that can immediately render the gaming experience non-functional.

Proof of Concept

```
// Current implementation allows problematic scenarios
function setSeasonTime(uint40 startTime, uint40 endTime) external
   onlyOwner {
    $.startTime = startTime;  // No validation - could be in the past
    $.endTime = endTime;      // No validation - could be before
       startTime
}

// Problematic scenarios:
// 1. Set past endTime -> season appears ended immediately
setSeasonTime(validStartTime, block.timestamp - 1 days);

// 2. Set endTime before startTime -> season never active
setSeasonTime(1000, 500);

// 3. Both in past with wrong order -> complete disruption
setSeasonTime(block.timestamp - 1 days, block.timestamp - 2 days);
```

Recommendation

Consider adding input validation following the pattern used in other OCH contracts:

```
function setSeasonTime(uint40 startTime, uint40 endTime) external
   onlyOwner {
    DragmaStorage storage $ = _getStorage();

    // Validate startTime
    if ($.startTime != startTime) {
        // Prevent changing startTime if season already started
        if (block.timestamp >= $.startTime && $.startTime != 0) {
            revert NotAllowed();
        }
        $.startTime = startTime;
    }

    // Validate endTime
    if ($.endTime != endTime) {
        // Ensure endTime is in future and after startTime
        if (block.timestamp >= $.endTime && $.endTime != 0) {
            revert NotAllowed();
        }
        if (endTime <= startTime) {
            revert NotAllowed();
        }
        $.endTime = endTime;
    }
}
```

The `Endgame.sol` contract already implements proper validation for similar functionality, demonstrating the awareness of this pattern:

solidity

```
// From Endgame.sol - proper validation example
if (block.timestamp >= endSeasonTime || _endTime < startSeasonTime) {
    revert NotAllowed();
}
```

Team Response

Acknowledged.

, our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!