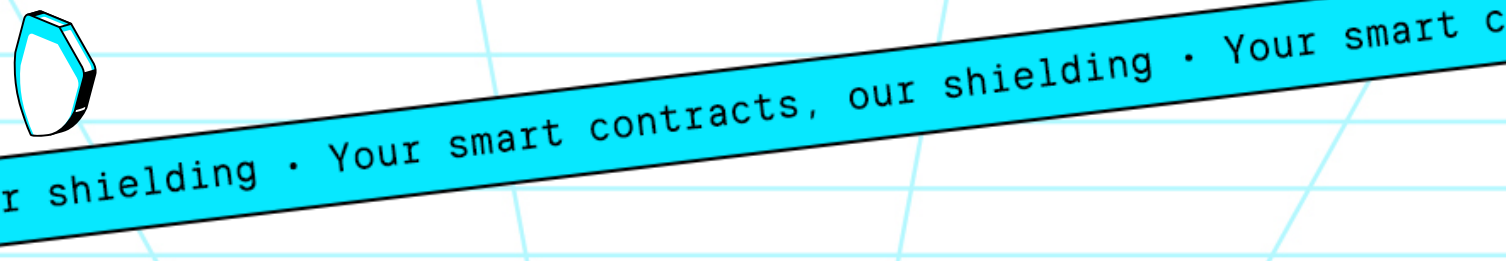
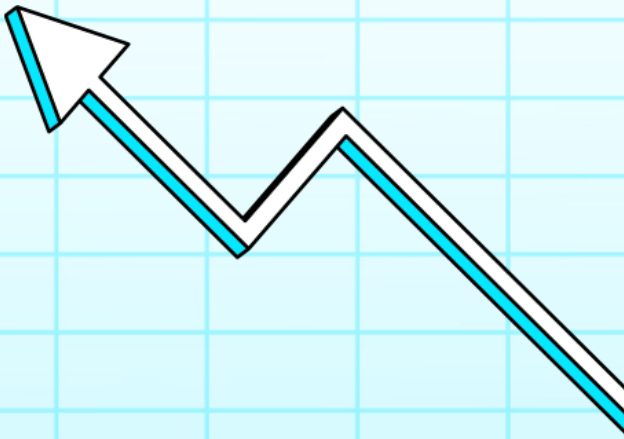


shieldify



Souls Club

Wheel

SECURITY REVIEW

Date: 20 November 2025

CONTENTS

1. **About Shieldify Security** 3
2. **Disclaimer** 3
3. **About Souls Club – Wheel** 3
4. **Risk classification** 4
   - 4.1 Impact 4
   - 4.2 Likelihood 4
5. **Security Review Summary** 4
   - 5.1 Protocol Summary 4
   - 5.2 Scope 4
6. **Findings Summary** 5
7. **Findings** 5

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 Souls Club - Wheel

Overview

A multiplayer game built around a simple concept: players place bets to grow their cell = chance. When the timer ends, a wheel spin decides the winner who takes the entire Bank — others lose. Sometimes, even the smallest bet takes it all. Players can view the full history and verify the transparent randomness of every game.

Basic rules

- Minimum bet: 0.005, maximum bet will be set based on player feedback.
- Commission: taken only from the winner, dynamic from 0.01% to 5% of the total Bank, always keeping the winner in profit. 1.69% of the Bank goes to the next season's raffle pool.

Dynamic commission

Each round has a pot — the sum of all bets. One player wins, the others lose. We compute: - O = total bets of the losers, - S = the winner's bet.

The commission depends on how balanced the bets were: - If the winner bet much more than the others, we take almost only from the losers; - If the bets were roughly equal, we take the full 5% from the entire pot.

Formula: commission = 5% × (O + min(O, S))

Examples: - Winner bet $300, others together $250 commission = 5% × (250 + 250) = $25 (4.5% of the $550 pot).

- Winner $300, others $300 commission = 5% × (300 + 300) = $30 (exactly 5% of the $600 pot).
- Winner $300, others $50 commission = 5% × (50 + 50) = $5 (1.6% of the $350 pot).

The closer the bets are to each other, the higher the commission. The more the winner "overcovered" everyone, the lower the commission.

4. Risk Classification

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

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 3 days, with a total of 24 hours dedicated by the Shieldify team.

Overall, the code is well-written. The audit report flagged one High, one Medium and three Low severity issues. They are mainly related to stake manipulation during cooldown and indefinitely trapped funds due to the single-player session lock.

The Souls Club team has done a great job with their test suite and provided exceptional support to the Shieldify researchers.

5.1 Protocol Summary

Project Name	Souls Club - Wheel
Repository	revolver-contract
Type of Project	GameFi, ETH Staking, VRF
Security Review Timeline	3 days
Review Commit Hash	cae64f2684c68204c07240f4815fa0fcbe0297cb
Fixes Review Commit Hash	890fba3f4d64c4d121c158367a5f81f320004dd6

5.2 Scope

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

File	nSLOC
contracts/Wheel.sol	246
Total	**246**

6. Findings Summary

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

- **High** issues: **1**
- **Medium** issues: **1**
- **Low** issues: **3**

ID	Title	Severity	Status
[H-01]	Stake Manipulation During Cooldown Period Allows Probability Gaming	High	Fixed
[M-01]	Single Player Session Lock – Funds Trapped Indefinitely	Medium	Fixed
[L-01]	Missing `_disableInitializers()` in Constructor	Low	Fixed
[L-02]	Session Status Not Updated to Finished After Winner Selection	Low	Fixed
[L-03]	Missing Validation Between `minStake` and `maxStake` Settings	Low	Fixed

7. Findings

[H-01] Stake Manipulation During Cooldown Period Allows Probability Gaming

Severity

High Risk

Description

The Wheel contract allows players to call `joinGame()` multiple times to increase their stake even after the cooldown period has begun. This creates a severe vulnerability where an attacker can manipulate their winning probability to near 100% by continuously depositing funds during the cooldown window.

Attack Scenario:

1. **Initial Setup**: Attacker deposits `minStake` (0.0000001 ETH) to join as the first player
2. **Trigger Cooldown**: A legitimate user deposits, triggering the 60-second cooldown (`timeoutDuration`)
3. **Cooldown Exploitation**: During the 60-second window, the attacker repeatedly calls `joinGame()`:
    - Each call can deposit up to `maxStake` (1 ETH)

- Total stake can reach: 60 transactions × 1 ETH = 60 ETH (or more with faster transactions)
- Meanwhile, a legitimate user has only a small stake (e.g., 0.01 ETH)

4. **Winner Selection**: Attacker now has 60 ETH out of 60.01 ETH total pool = 99.98% win probability

Example Calculation:

```
Legitimate User Stake: 0.01 ETH (0.02% probability)
Attacker Initial Stake: 0.0000001 ETH
Attacker During Cooldown: 60 ETH (via 60 deposits of maxStake)
Total Pool: 60.0100001 ETH
Attacker Win Probability: (60.0000001 / 60.0100001) × 100 = 99.98%
```

Location of Affected Code

File: contracts/Wheel.sol#L295-L360

```
function joinGame(
    string calldata uid,
    bytes calldata signature
) external payable nonReentrant {
    Session storage session = sessions[currentSessionId];
    if (session.status != SessionStatus.Gathering)
        revert SessionNotGathering();
    if (msg.value < minStake) revert StakeTooLow();
    // @audit can deposit multiple times to increase during cooldown
    if (msg.value > maxStake) revert StakeTooHigh();

    // ... signature verification ...

    // Track unique participants and pool
    uint256 previousStake = session.stakeByPlayer[msg.sender];
    if (previousStake == 0) {
        session.uniquePlayersCount++;
        session.participants.push(msg.sender);
        if (session.participants.length > MAX_PARTICIPANTS) revert
           MaxParticipantsExceeded();
    }

    session.stakeByPlayer[msg.sender] = previousStake + msg.value; //
       Unlimited accumulation
    session.pool += msg.value;

    // Activate when there are at least two unique players
    // @audit what if only 1 user joins?
    if (session.uniquePlayersCount >= 2 && session.startTimestamp == 0) {
        session.startTimestamp = block.timestamp; // Cooldown starts
        emit SessionActivated(currentSessionId, session.startTimestamp);
    }
}
```

File: contracts/Wheel.sol#L365-L378

```
function startSession(uint256 traceId) external onlyBackend {
    Session storage session = sessions[currentSessionId];
    if (session.status != SessionStatus.Gathering) revert InvalidState();
    if (session.uniquePlayersCount < 2) revert InvalidState();
    // @audit why cooldown period is given?
    if (block.timestamp < session.startTimestamp + timeoutDuration)
        revert WaitingForTimeout(); // 60 seconds of exploitation window
    // ...
}
```

Impact

The attacker can achieve near-guaranteed wins (99%+ probability) in every session. The game becomes entirely unfair and predictable for the attacker.

Recommendation

Implement a comprehensive stake cap system that limits any single player's control over the total pool.

Team Response

Fixed.

[M-01] Single Player Session Lock - Funds Trapped Indefinitely

Severity

Medium Risk

Description

The Wheel contract allows a single player to join a session and deposit funds, but the session can never be completed with only one participant. There is no mechanism for a solo player to withdraw their funds or cancel their participation, resulting in a permanent fund lock until a second player joins. This creates a critical user experience issue and a potential fund lock scenario.

The Problem Chain:

```
Player 1 deposits 1 ETH → `uniquePlayersCount = 1`
`startTimestamp` remains 0 (never set)
`startSession()` cannot be called (requires `uniquePlayersCount >= 2`)
Session never completes
Player 1 cannot claim funds (no prize balance)
Player 1 cannot withdraw (no withdrawal function)
Funds are locked until a second player joins
```

Location of Affected Code

File: contracts/Wheel.sol#L295-L360

```
function joinGame( string calldata uid, bytes calldata signature )
   external payable nonReentrant {
    Session storage session = sessions[currentSessionId];
    if (session.status != SessionStatus.Gathering)
        revert SessionNotGathering();
    if (msg.value < minStake) revert StakeTooLow();
    if (msg.value > maxStake) revert StakeTooHigh();

    // ... signature verification ...

    uint256 previousStake = session.stakeByPlayer[msg.sender];
    if (previousStake == 0) {
        session.uniquePlayersCount++;
        session.participants.push(msg.sender);
        if (session.participants.length > MAX_PARTICIPANTS) revert
           MaxParticipantsExceeded();
    }

    session.stakeByPlayer[msg.sender] = previousStake + msg.value;
    session.pool += msg.value;

    emit GameJoined(currentSessionId, msg.sender, uid, session.
       stakeByPlayer[msg.sender]);

    // Activate when there are at least two unique players
    // @audit what if only 1 user joins?
    if (session.uniquePlayersCount >= 2 && session.startTimestamp == 0) {
        session.startTimestamp = block.timestamp;
        emit SessionActivated(currentSessionId, session.startTimestamp);
    }
    //   If uniquePlayersCount == 1, session.startTimestamp remains 0
       forever
}
```

File: contracts/Wheel.sol#L365-L378

```
function startSession(uint256 traceId) external onlyBackend {
    Session storage session = sessions[currentSessionId];
    if (session.status != SessionStatus.Gathering) revert InvalidState();
    if (session.uniquePlayersCount < 2) revert InvalidState(); //
       Blocks single player sessions
    if (block.timestamp < session.startTimestamp + timeoutDuration)
        revert WaitingForTimeout();
    // code
}
```

Impact

If a single player joins and no second player ever arrives:

- Player's funds are locked in the contract indefinitely.
- No mechanism to withdraw or cancel participation.

- No refund functionality exists.
- Funds are effectively lost until another player joins.

Recommendation

Players must be able to withdraw their stake after a reasonable waiting period if the session fails to attract the minimum required participants.

Team Response

Fixed.

[L-01] Missing `_disableInitializers()` in Constructor

Severity

Low Risk

Description

The Wheel contract is a UUPS upgradeable contract that inherits from OpenZeppelin's Initializable and UUPSUpgradeable contracts. However, it does not include a constructor that calls `_disableInitializers()` to protect the implementation contract from being initialized. Without this constructor, an attacker can call initialize() directly on the implementation contract (not the proxy), potentially causing unexpected behavior or security issues.

Location of Affected Code

File: contracts/Wheel.sol#L262-L279

```
function initialize(
    address _vrfSystem,
    address _admin,
    address _backend
) external initializer {
      __UUPSUpgradeable_init();
      __ReentrancyGuard_init();
      vrfSystem = IVRFSystem(_vrfSystem);
      admin = _admin;
      backend = _backend;
      currentSessionId = 1;
      sessions[currentSessionId].status = SessionStatus.Gathering;
      minStake = 0.0000001 ether;
      maxStake = 1 ether;
      commissionPercent = 500;
      timeoutDuration = 60;
      MAX_PARTICIPANTS = 10000;
}
```

Impact

An attacker could initialize the implementation contract directly, setting themselves as the admin and backend addresses.

Recommendation

Add a constructor that calls `_disableInitializers()` to prevent the implementation contract from being initialized.

Team Response

Fixed.

[L-02] Session Status Not Updated to Finished After Winner Selection

Severity

Low Risk

Description

The `randomNumberCallback()` function successfully completes the winner selection process, distributes prizes, and advances to the next session, but fails to update the current session's status to `SessionStatus.Finished`. This leaves the completed session in the `SessionStatus.WaitingVRF` state permanently, creating inconsistencies in session state management and potentially breaking dependent functionality.

Location of Affected Code

File: contracts/Wheel.sol#L383-L425

```
function randomNumberCallback(
    uint256 requestId,
    uint256 randomNumber
) external onlyVRFSystem {
    uint256 sessionId = vrfRequestToSessionId[requestId];
    Session storage session = sessions[sessionId];
    if (session.status != SessionStatus.WaitingVRF) revert InvalidState()
        ;

    // Weighted random selection by stake
    uint256 totalStake = session.pool;
    require(totalStake > 0, "No stake");
    uint256 target = randomNumber % totalStake;
    address winner = address(0);

    uint256 len = session.participants.length;
    for (uint256 i = 0; i < len; ++i) {
        address p = session.participants[i];
        uint256 s = session.stakeByPlayer[p];
        if (s == 0) continue;
        if (target < s) {
            winner = p;
            break;
        }
        target -= s;
    }

    // Fallback safety (should not happen)
    if (winner == address(0)) {
        winner = session.participants[len - 1];
    }

    uint256 sWinner = session.stakeByPlayer[winner];
    (uint256 commission, ) = previewDynamicCommission(totalStake, sWinner
        , commissionPercent);

    accumulatedCommission += commission;
    uint256 payoutPool = totalStake - commission;

    prizeBalance[winner] += payoutPool;

    emit GameFinished(sessionId, winner, payoutPool, commission,
        requestId);
    // @audit status not set to finish  Missing: session.status =
        SessionStatus.Finished;

    _advanceSession();
}
```

Impact

Historical sessions remain in the `WaitingVRF` state even after completion.

Recommendation

Add the missing status update before advancing to the next session.

Team Response

Fixed.

[L-03] Missing Validation Between `minStake` and `maxStake` Settings

Severity

Low Risk

Description

The `setMinStake()` and `setMaxStake()` functions allow the admin to independently update the minimum and maximum stake values without validating their relationship. This can result in a state where `minStake > maxStake`, making it impossible for any player to join the game and effectively bricking the contract's core functionality.

Location of Affected Code

File: contracts/Wheel.sol#L482-L491

```
/**
 * @notice Updates the minimum stake amount required to join a game
 * @dev Can only be called by the admin
 *      Cannot be set to zero
 * @param _newMinStake The new minimum stake amount in wei
 */
// @audit should check for this minStake > maxStake
function setMinStake(uint256 _newMinStake) external onlyAdmin {
    if (_newMinStake == 0) revert InvalidState();
    minStake = _newMinStake;  //  No validation against maxStake
}
```

File: contracts/Wheel.sol#L493-L502

```
/**
 * @notice Updates the maximum stake amount allowed to join a game
 * @dev Can only be called by the admin
 *      Cannot be set to zero
 * @param _newMaxStake The new maximum stake amount in wei
 */
function setMaxStake(uint256 _newMaxStake) external onlyAdmin {
    if (_newMaxStake == 0) revert InvalidState();
    maxStake = _newMaxStake;  //  No validation against minStake
}
```

Impact

If `minStake > maxStake`, no player can join any game session.

Recommendation

Add Cross-Validation in Both Functions.

Team Response

Fixed.

shieldify

Thank you!