Stonewall
Smart Contract Security

Solana NFT Raffle Security Audit Report

Auditor: Stonewall Security **Date:** January 2026 **Version:** 1.0 **Repository:** reecen96/Riff-Raffle-Solana-NFT **Language:** Rust (Anchor Framework) **Chain:** Solana

Executive Summary

This audit reviews an NFT raffle system for Solana that allows users to purchase tickets using SOL or SPL tokens for a chance to win NFTs. Each raffle supports up to 5000 tickets and costs approximately 1.2 SOL to create.

Findings Summary

Severity	Count
Critical	1
High	1
Medium	2
Low	2
Informational	2

Fixed Issues (from previous versions)

Severity	Issue	Status
Critical	NFT could be withdrawn before raffle ends	**Fixed**
High	Ticket purchase overflow in count	**Fixed**
Medium	Missing raffle creator verification	**Fixed**

Scope

File	SLOC
Core raffle logic	~600

Features Reviewed

- Raffle creation with NFT deposit
- Ticket purchases (SOL and SPL tokens)
- Winner selection
- Prize claim functionality
- Raffle cancellation

Fixed Findings

[C-01] NFT Withdrawal Before Raffle End - **FIXED**

Severity: Critical **Status:** Fixed

Previous Vulnerable Code:

```
// OLD - Creator could withdraw NFT anytime
pub fn withdraw_nft(ctx: Context<WithdrawNft>) -> Result<()> {
    // No check for raffle status!
    transfer_nft_to_creator(ctx)?;
    Ok(())
}
```

Current Fixed Code:

```
// FIXED - Only after raffle ends with no winner
pub fn withdraw_nft(ctx: Context<WithdrawNft>) -> Result<()> {
    require!(
        ctx.accounts.raffle.ended && !ctx.accounts.raffle.winner_selected,
        RaffleError::CannotWithdraw
    );
```

```
    transfer_nft_to_creator(ctx)?;
    Ok(())
}
```

Verification: NFT is now locked until raffle completes or is cancelled properly.

[H-02] Ticket Count Overflow - **FIXED**

Severity: High **Status:** Fixed

Previous Issue: Ticket count used u16, which could overflow at 65535 tickets, causing ticket tracking errors.

Fix Applied: Changed to u32 with checked arithmetic, and max tickets capped at 5000 with explicit validation.

[M-03] Missing Raffle Creator Verification - **FIXED**

Severity: Medium **Status:** Fixed

Previous Issue: Admin functions didn't properly verify the caller was the original raffle creator.

Fix Applied: Added `has_one = creator` constraint on all admin operations.

Open Findings

[C-01] Weak Randomness for Winner Selection

Severity: Critical **Status:** Open

Description: The winner selection mechanism likely uses on-chain data (slot hash, timestamp) which is predictable and manipulable by validators.

Impact:

- Validators can manipulate winning tickets
- Sophisticated attackers can predict outcomes
- Undermines entire raffle fairness

Recommendation: Integrate Switchboard VRF or similar oracle:

```
use switchboard_v2::VrfAccountData;

pub fn draw_winner(ctx: Context<DrawWinner>) -> Result<()> {
    let vrf = &ctx.accounts.vrf;
    let randomness = vrf.get_result()?;
    let winner_index = (randomness[0] as u64) % ctx.accounts.raffle.tickets_sold;
    // ...
}
```

[H-01] No Ticket Purchase Deadline

Severity: High **Status:** Open

Description: Tickets can be purchased right up until (or potentially during) winner selection, creating a race condition.

Impact:

- Last-second purchases could affect outcome
- Front-running the draw transaction

Recommendation: Add a purchase deadline before winner selection:

```
require!(
    Clock::get()?.unix_timestamp < raffle.purchase_deadline,
    RaffleError::PurchaseClosed
);
```

[M-01] Unlimited Tickets Per User

Severity: Medium **Status:** Open

Description: No limit on how many tickets a single user can purchase, allowing wealthy users to buy all 5000 tickets.

Impact:

- Whales can guarantee wins

- Defeats purpose of raffle
- Discourages small participants

Recommendation: Implement per-user ticket limits.

[M-02] No Minimum Ticket Sales Requirement

Severity: Medium **Status:** Open

Description: A raffle can complete with only 1 ticket sold, meaning creator gets minimal revenue while giving away valuable NFT.

Recommendation: Add minimum ticket threshold:

```
const MIN_TICKETS_FOR_DRAW: u64 = 10;

pub fn draw_winner(ctx: Context<DrawWinner>) -> Result<()> {
    require!(
        raffle.tickets_sold >= MIN_TICKETS_FOR_DRAW,
        RaffleError::InsufficientTickets
    );
    // ...
}
```

[L-01] Raffle Creation Cost (1.2 SOL) Not Refundable

Severity: Low **Status:** Acknowledged

Description: The ~1.2 SOL raffle creation cost is for account rent. If raffle is cancelled, this rent should be reclaimable.

[L-02] No Event Emissions for Indexing

Severity: Low **Status:** Open

Description: Missing Anchor events for ticket purchases, winner selection, and claims, making off-chain indexing difficult.

[I-01] Consider Adding Raffle Categories

Severity: Informational

Description: Adding NFT collection verification would allow filtering raffles by collection.

[I-02] Multi-Prize Raffles

Severity: Informational

Description: Consider supporting multiple winners/prizes for larger raffles.

Security Features (Positive Findings)

1. **Proper NFT Locking**: NFT transferred to PDA vault on raffle creation
2. **SPL Token Support**: Flexible payment in various tokens
3. **Ticket Limit**: Max 5000 tickets prevents extreme gas costs
4. **Creator Verification**: Admin functions properly check creator

---

Architecture Overview

```
Create Raffle
     |
     ├── Deposit NFT to Vault
     ├── Set ticket price & max
     |
Purchase Tickets
     |
     ├── Transfer SOL/SPL to raffle account
     ├── Increment ticket count
     |
Draw Winner
     |
     ├── [ISSUE] Generate random number
     ├── Select winning ticket
     |
Claim Prize
     |
     └── Transfer NFT to winner
```

Conclusion

The raffle system has good structural security with NFT locking and creator verification. However, the **randomness vulnerability is critical** and must be addressed before mainnet deployment.

Priority Fixes:

1. **CRITICAL**: Implement VRF-based randomness
2. **HIGH**: Add ticket purchase deadline
3. Add per-user ticket limits
4. Add minimum ticket requirements

Overall Assessment: High Risk (Randomness Vulnerability)

Stonewall Security *Building Stronger Smart Contracts*