IRONNODE

ironnode.io

MY CRYPTO CASINO

Solana Program Security Audit

by **IronNode** from **Jan 09 - Jan 25, 2025**

Executive overview

1 Executive overview

1.1 Introduction

MyCryptoCasino is an Anchor-based Solana program implementing a crash gambling game with:

- USDC-denominated betting
- Commit-reveal scheme for provably fair crash values
- NFT-based multi-level referral system (5 ancestor levels)
- Per-round and per-bet profit caps to limit exposure

1.2 Audit summary

Our security audit team was allocated time to conduct an extensive review of the **MyCryptoCasino** Solana contract. The audit was led by experienced security engineers with expertise in blockchain technology, smart contract security, and Rust programming.

1.2.1 Objectives of the audit

1. Examine all instruction handlers for common Solana-specific vulnerabilities including missing signer checks, improper account validation, PDA seed collisions, arithmetic overflow/underflow, and reentrancy issues.
2. Analyze the commit-reveal scheme for the crash value to ensure the casino cannot manipulate outcomes after bets are placed, and that players cannot predict or influence crash values.
3. Evaluate all fund flows (deposits, withdrawals, bet placement, payouts, referral rewards) to ensure user and casino funds cannot be stolen, locked, or drained through malicious transactions.
4. Verify that authority checks are properly enforced across all privileged operations including casino management, game administration, and configuration updates.
5. Assess the NFT-based referral mechanism for potential abuse vectors such as self-referral exploits, reward manipulation, or improper reward distribution.
6. Ensure the crash game lifecycle cannot be manipulated to bypass intended game rules or create inconsistent state.
7. Verify that crash value calculations, payout multipliers, and fee distributions handle edge cases correctly without precision loss or rounding exploits.

1.3 Scope

1.3.1 Code repositories

Branch: develop

Commit Hash: 096d23c744ff139711111a34ad490876d0e2ee67

Remediation Commit ID

255261c986c864b1eda1c9bef1a4193e54e8d1d1

1.4 Assessment summary & findings overview

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	2

Security analysis	Risk Level	Remediation
Multiple Account Creation Bypasses Referral NFT Mint Fee Leading To Revenue Loss	MEDIUM	ACCEPTED
Unchecked unwrap() Calls On Arithmetic Operations Lead To Potential Program Panic	LOW	FIXED
Missing Mint Account Type Validation in update_config() Allows Setting Arbitrary Pubkey as USDC Mint	LOW	FIXED
PDA Bump Seeds Not Stored in Account State Leading to Unnecessary Compute Unit Consumption	LOW	FIXED
Game State Functions Lack Explicit PDA Seed Validation	INFORMATIONAL	FIXED
Absence of Event Emissions in Deposit and Withdrawal Functions Limits Off-Chain Observability	INFORMATIONAL	FIXED

1.5 Remediation Notes

We reviewed the client's remediation changes at commit 255261c986c864b1eda1c9bef1a4193e54e8d1d1 and re-validated all the entries listed in Section 2 of this report. At this commit, all issues appear fully fixed, and 1/21 are partially fixed.

- **Multiple Account Creation Bypasses Referral NFT Mint Fee Leading To Revenue Loss (Accepted):** Client acknowledged this issue as working as designed.
- **Unchecked** unwrap() **Calls On Arithmetic Operations Lead To Potential Program Panic (Fixed):** Client fixed this issue by replacing all unwrap() calls on checked arithmetic operations with proper Anchor error propagation using explicit errors across all instruction handlers.
- **Missing Mint Account Type Validation in** update_config() **Allows Setting Arbitrary Pubkey as USDC Mint (Fixed):** Client fixed this issue by replacing the raw Pubkey parameter with Option<Account<'info, Mint>>, leveraging Anchor's automatic account deserialization to validate the provided account is a valid SPL Token Mint before execution.
- **PDA Bump Seeds Not Stored in Account State Leading to Unnecessary Compute Unit Consumption (Fixed):** Client fixed this issue by adding a bump field to all 9 PDA-derived state accounts, persisting bumps at initialization, and replacing bare bump constraints with stored references across all instructions.
- **Game State Functions Lack Explicit PDA Seed Validation (Fixed):** Client fixed this issue by adding explicit PDA seed and bump constraints to previously unconstrained accounts in cancel_game, start_game, cancel_bet, and claim_rewards instructions, ensuring only legitimate PDA-derived accounts are accepted.
- **Absence of Event Emissions in Deposit and Withdrawal Functions Limits Off-Chain Observability (Fixed):** Client fixed this issue by adding CasinoDeposited, CasinoWithdrawn, UserDeposited and UserWithdrawn event structs and emitting them in all four deposit/withdraw instruction handlers with version, authority, amount, and new balance fields.

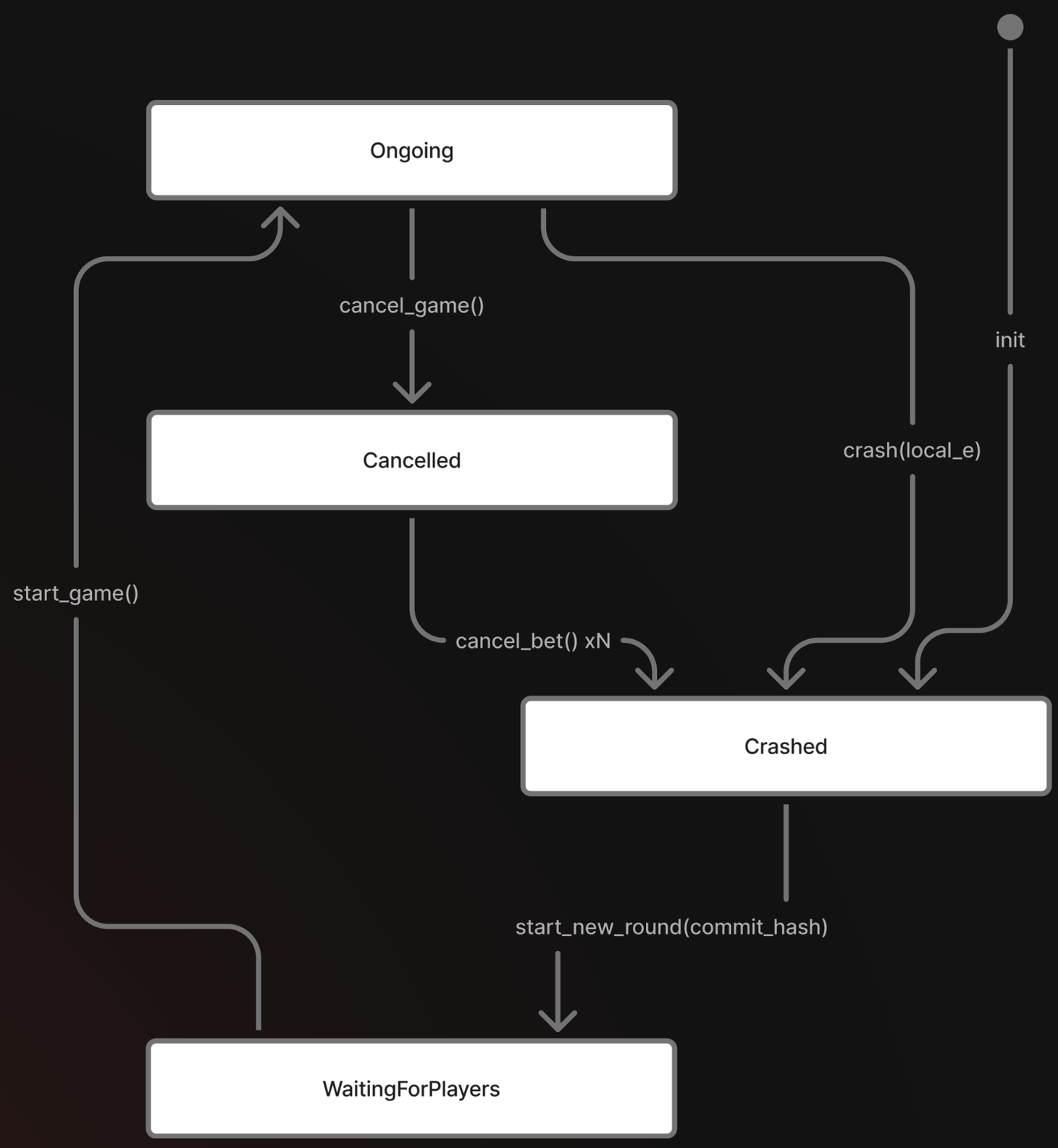
Application Flow Analysis

4 Application Flow Analysis

This section of the report provides an analysis of the application flow for the **MyCryptoCasino** contract. The analysis covers the logical flow of the code. With the trust boundaries:

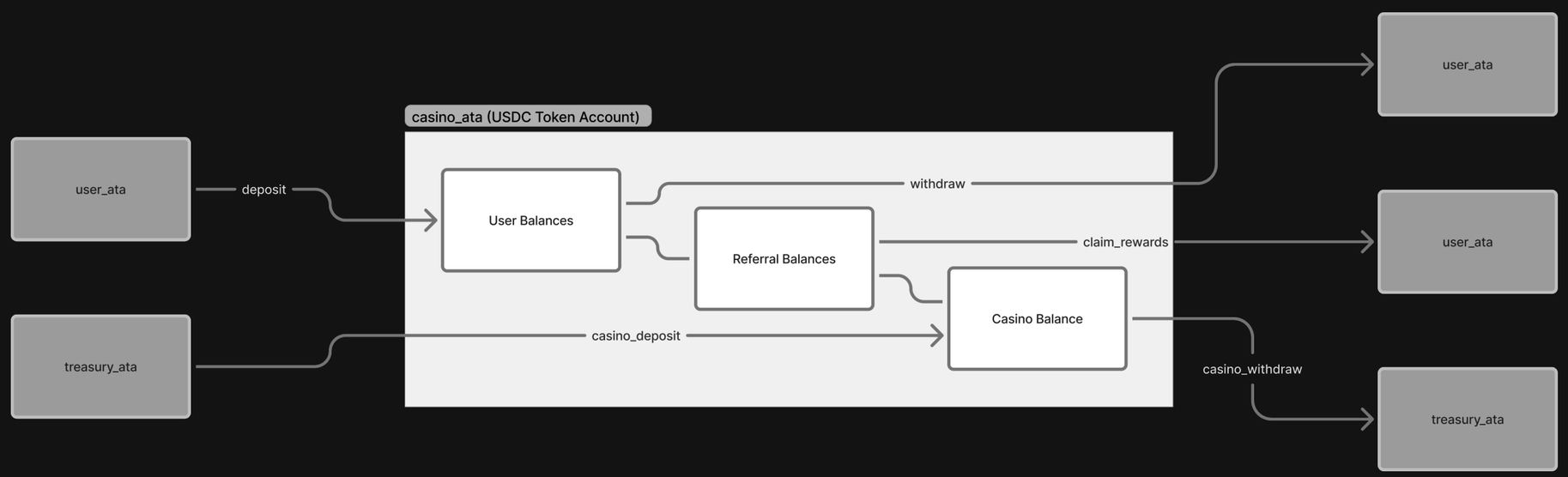
1. config_authority > treasury_authority: Separate roles. Config cannot directly move funds, treasury cannot change config.
2. config_authority > game_authority: Config can override game_authority (emergency), toggle lock, but cannot run rounds.
3. game_authority: Has significant power (all game operations, placing bets for users). Compromise means full round manipulation.
4. Players: No direct token movements during betting; only virtual user balance changes. Real tokens move only on deposit/withdraw.

2.1 Game State Flow



2.2 Fund Flows

2.2.1 Casino Token Account Flow



2.2.1 Virtual Token Flow



Findings

4 Findings

4.1 Multiple Account Creation Bypasses Referral NFT Mint Fee Leading To Revenue Loss

The referral system allows users to mint NFT-based referral codes. The first mint per user is free, while subsequent mints require payment of mint_price_lamports to the casino treasury. This design intends to balance user onboarding (free first code) with spam prevention (paid additional codes). The User account tracks mints via the referral_nfts_minted counter, and the max_referral_nfts_per_user configuration limits total mints per user.

The create_referral() function in CreateReferral checks whether to charge the mint fee based on the calling user's mint count:

The User account is a PDA seeded with authority.key(), meaning each wallet address has an independent user account with its own referral_nfts_minted counter. Combined with the transferability of Metaplex Core NFTs, this creates a bypass.

The claim_rewards() function validates ownership via the current NFT owner, not the original creator, so transferred NFTs retain full functionality.

4.1.1 Code location

src/instructions/referral/create_referral.rs

```
let mint_price = nft_config.mint_price_lamports;
if user.referral_nfts_minted > 0 && mint_price > 0 {
    let transfer_accounts = Transfer {
        from: ctx.accounts.payer.to_account_info(),
        to: treasury_authority.to_account_info(),
    };
    let cpi_ctx = CpiContext::new(
        ctx.accounts.system_program.to_account_info(),
        transfer_accounts,
    );
    transfer(cpi_ctx, mint_price)?;
}
```

4.1.2 Impact

The casino treasury loses expected revenue from referral NFT mints. The magnitude depends on:

- The configured mint_price_lamports value
- The number of users exploiting this bypass

The max_referral_nfts_per_user limit is also bypassed, as each created wallet has an independent counter. A single entity can accumulate unlimited referral NFTs.

4.1.3 Recommendation

Several mitigation strategy examples could be:

- Maintain a global or per-wallet counter that increments when an NFT is transferred to an address, not just when minted. This requires hooking into transfer events or using a separate registry.
- Require users to have an account age, minimum balance, or staked amount before minting referral NFTs.
- Charge mint_price_lamports for all mints or consider removing the mint price
- If the mint fee is primarily symbolic and the business model does not depend on this revenue, document the behavior as intended.

4.2 Unchecked unwrap() Calls On Arithmetic Operations Lead To Potential Program Panic

Multiple functions throughout the codebase use unwrap() on Option<u64> values returned from operations. While checked arithmetic methods prevent silent overflow by returning None on arithmetic errors, calling unwrap() on a None value causes the program to panic, halting execution without returning a proper Anchor error. This pattern appears in 32 locations across 14 instruction files.

4.2.1 Impact

The use of unwrap() violates graceful error handling principles:

- A panic produces an unhelpful error message instead of a descriptive Anchor error
- Program behavior under edge cases becomes unpredictable
- It complicates debugging and monitoring

4.2.2 Recommendation

- Replace all unwrap() calls with proper error handling using ok_or() or ok_or_else()
- Define dedicated error variants in the errors module.

4.3 Missing Mint Account Type Validation In Update Config Allows Setting Arbitrary Pubkey As USDC Mint

The update_config() instruction allows the config_authority to update the USDC mint address stored in the global Config account. This mint address is critical as it determines which token the protocol accepts for all betting and payout operations.

The new_usdc_mint parameter is accepted as a raw Pubkey passed through instruction data rather than as a validated account in the UpdateConfig struct. Anchor performs no verification that the provided address corresponds to an actual SPL Token Mint account. Any 32-byte value can be stored as the USDC mint.

4.3.1 Code location

src/instructions/config/update_config.rs

```
pub fn update_config(ctx: Context<UpdateConfig>, new_usdc_mint: Option<Pubkey>) -> Result<()> {
    let config = &mut ctx.accounts.config;

    require!(
        ctx.accounts.authority.key() == config.config_authority,
        AdminError::NotAllowed
    );

    if let Some(usdc) = new_usdc_mint {
        config.usdc_mint = usdc;
    }

    Ok(())
}
```

4.3.2 Impact

If the config_authority mistakenly sets an invalid pubkey (typo, wrong address, or non-mint account), subsequent protocol operations that interact with the USDC mint will fail. While the trusted admin controls this function and can correct the mistake, the lack of validation creates an opportunity for configuration errors that could temporarily disrupt protocol functionality.

4.3.4 Recommendation

Pass the mint as an account in the UpdateConfig struct with proper type validation.

4.4 PDA Bump Seeds Not Stored In Account State Leading To Unnecessary Compute Unit Consumption

Program Derived Addresses (PDAs) in Solana are generated using a canonical bump seed that ensures the address does not lie on the ed25519 curve. The find_program_address() function iterates from bump value 255 downward until a valid off-curve address is found.

None of the state accounts store a bump field. Instead, every instruction that validates or uses these PDAs relies on Anchor's bump constraint which recalculates the canonical bump at runtime, e.g.:

```
#[account(mut, seeds = [CASINO_SEED], bump)]
pub casino: Account<'info, Casino>,
```

The bump recalculation occurs across 30+ instruction account constraints throughout the codebase.

4.4.2 Impact

The compute unit overhead depends on the canonical bump value for each PDA. For a typical transaction involving multiple PDAs (e.g., payout() which validates User, CrashGame, Casino, and Bet), this overhead compounds. While individual transactions remain within Solana's 1.4M CU limit, unnecessary CU consumption:

- Increases transaction priority fees for users during network congestion
- Reduces headroom for complex transactions
- Represents suboptimal resource utilization

4.4.3 Recommendation

- Store the canonical bump in each PDA account during initialization and use bump = <account>.bump for validation.

4.5 Absence Of Event Emissions In Deposit And Withdrawal Functions Limits Off-Chain Observability

The deposit(), casino_deposit(), withdraw() and casino_withdraw() instructions in the user and config modules handle balance changes for player accounts. These functions update the User and Casino account state to reflect new balances after deposits or withdrawals.

However, neither function emits events upon successful execution. Solana programs can emit events using the emit! macro, which allows off-chain indexers, analytics tools, and frontend applications to track state changes without polling account data directly.

4.5.2 Impact

Without event emissions:

- Off-chain indexers must rely on account polling or transaction parsing to detect balance changes
- Building real-time dashboards or notifications for deposit/withdrawal activity becomes more complex

This does not affect on-chain security or correctness of the protocol.

4.5.3 Recommendation

Consider adding the missing event emissions to improve off-chain observability.

4.6 Game State Functions Lack Explicit PDA Seed Validation

The crash game lifecycle includes administrative functions that transition game state. The cancel_game() and start_game() functions validate authority through game_authority checks but do not enforce explicit PDA seed derivation constraints on the crash_game account within their Anchor account validation structs.

Both functions rely solely on the game_authority signer check for access control. While Anchor's account deserialization verifies the discriminator, explicit PDA seed validation is absent.

4.6.1 Code location

src/instructions/crash_game/cancel_game.rs

```
#[derive(Accounts)]
pub struct CancelGame<'info> {
    #[account(mut)]
    pub crash_game: Account<'info, CrashGame>,
    pub game_authority: Signer<'info>,
    // No seeds constraint on crash_game
}
```

src/instructions/crash_game/start_game.rs

```
#[derive(Accounts)]
pub struct StartGame<'info> {
    #[account(mut)]
    pub crash_game: Account<'info, CrashGame>,
    pub game_authority: Signer<'info>,
    // No seeds constraint on crash_game
}
```

4.6.2 Impact

No security impact exists. Both functions are restricted to the game_authority signer, and no fund transfers occur within these instructions. The crash_game account type is validated through Anchor's discriminator check, preventing arbitrary account substitution.

4.6.3 Recommendation

Consider adding explicit PDA seed constraints for defense-in-depth and operational clarity.

IRONNODE

ironnode.io

Thank you for choosing us!