ZEALYNX

Web3 Security

Fair Casino

Solana Smart Contract Audit

January 19, 2026

contact@zealynx.io

Carlos (Bloqarl) @TheBlockChainer

Stephen @derastephh

Contents

1. **About Zealynx**
2. **Disclaimer**
3. **Overview**
   - 3.1 Project Summary
   - 3.2 About Fair Casino
   - 3.3 Audit Scope
4. **Audit Methodology**
5. **Severity Classification**
6. **Executive Summary**
7. **Audit Findings**
   - 7.1 High Severity Findings
   - 7.2 Medium Severity Findings
   - 7.3 Low Severity Findings

1. About Zealynx

Zealynx, founded in January 2024 by Carlos (Bloqarl), specializes in smart contract audits, and development. Our services include comprehensive smart contract audits, application security audits, such as pentesting, and AI Audits. We are trusted by clients such as Badger DAO, Ample protocol, Lido, Inverter, Matchain, and Golden Grid.

Our team believes in transparency and actively contributes to the community by creating educational content. This includes challenges for Foundry and Rust, security tips for Solidity developers, and fuzzing materials like Foundry and Echidna/Medusa. We also publish articles on topics such as preparing for an audit and battle testing smart contracts on our website Zealynx.io and our blogs.

Zealynx has achieved public recognition, including winning 1st prize in the Uniswap Hook Hackathon and a Top 5 position in the Beanstalk Audit public contest. Our ongoing commitment is to provide value through our expertise and innovative security solutions for smart contracts.

2. Disclaimer

A security review of a smart contract can't guarantee there are no vulnerabilities. It's a limited effort in terms of time, resources, and expertise to find as many vulnerabilities as possible. We can not assure 100% security post-review or guarantee the discovery of any issues with your smart contracts.

3. Overview

3.1 Project Summary

A time-boxed independent Solana smart contract audit of the Fair Casino protocol was conducted by Zealynx, with a focus on the potential security vulnerabilities.

We performed the security evaluation based on the agreed scope during the 2-week Solana + TypeScript audit, following our systematic approach and methodology. Based on the scope and our performed activities, our security assessment revealed 1 High, 1 Medium and 3 Low.

3.2 About Fair Casino

Fair Casino is an online gaming platform that operates a Solana-based vault program for secure token custody and withdrawal management. The protocol implements a non-custodial architecture where user deposits are held in a Program Derived Address (PDA) controlled vault, enabling the platform to process authorized withdrawals through cryptographic verification while maintaining separation of concerns between vault authority and withdrawal signing privileges.

The program validates backend-authorized withdrawals using Ed25519 signature verification via instruction introspection. Security features include replay protection through PDA-based withdrawal tracking, timestamp-based authorization expiry, emergency pause functionality, and two-step authority transfers. The vault is designed to manage FAIR token operations with administrative controls for operational security and incident response.

3.3 Audit Scope

The code under review is a Solana program written in Rust and includes ~280 nSLOC. The codebase consists of a single program file lib.rs implementing the casino vault architecture with PDA-based token custody, Ed25519 signature verification for withdrawal authorization, administrative controls for vault authority and withdrawal signer management, emergency pause functionality, and comprehensive replay protection mechanisms for withdrawal processing with timestamp-based expiry validation.

4. Audit Methodology

Approach

During our security assessments, we uphold a rigorous approach to maintain high-quality standards. Our methodology encompasses thorough **Manual Security Review**, **Solana-Specific Pattern Analysis**, and **Architecture Validation** focused on Program Derived Addresses, Cross-Program Invocations, and account constraint verification.

Throughout the smart contract audit process, we prioritize the following aspects to uphold excellence:

1. **Code Quality**: We diligently evaluate the overall quality of the code, aiming to identify any potential vulnerabilities or weaknesses.
2. **Best Practices**: Our assessments emphasize adherence to established best practices, ensuring that the smart contract follows industry-accepted guidelines and standards.
3. **Documentation and Comments**: We meticulously review code documentation and comments to ensure they accurately reflect the underlying logic and expected behavior of the contract.

5. Severity Classification

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

6. Executive Summary

This report covers the Solana vault program component audited as part of the Fair Casino security engagement. The Zealynx team conducted a focused security review of the on-chain program implementation.

The audit focused on identifying vulnerabilities and implementation issues affecting token custody, withdrawal authorization, signature verification, and access control mechanisms. Special attention was given to PDA security, Ed25519 instruction introspection, and administrative controls.

A total of 5 issues were identified and categorized as follows:
- 1 High severity
- 1 Medium severity
- 3 Low severity

The initial commit hash audited is:
- 955864f390f5599bd2fcbc3702fe460dd354efed

The key findings include:

- **Missing vault initialization authorization enables complete takeover**: The initialize_vault function does not validate the caller's identity, allowing anyone to become the vault authority. Since Solana programs require manual initialization, an attacker who frontruns the legitimate deployment can gain complete control over the vault including all administrative operations and user deposits with no recovery mechanism.
- **Missing token account constraints leads to potential unauthorized transfers**: The ProcessWithdrawal instruction context fails to validate critical properties of the token accounts used in withdrawal operations. Specifically, vault_token_account and recipient_token_account lack constraints verifying ownership and mint matching, creating opportunities for confusion attacks and potential transfers of unintended token types.
- **Single-step authority transfer risks permanent loss of control**: The vault implements authority transfer functions that immediately transfer control in a single transaction without requiring confirmation from the new authority. A single typographical error during authority transfer can result in permanent loss of vault control and all deposited funds with no recovery mechanism.

Summary of Findings :

Vulnerability	Severity	Status
[H-01] Missing caller authorization in initialize_vault allows initialization frontrunning leading to complete vault takeover	High	Fixed
[M-01] Missing token account constraints leads to potential mint confusion and unauthorized token transfers	Medium	Fixed
[L-01] Single-step authority transfer without confirmation risks permanent loss of vault control on operator error	Low	Fixed
[L-02] Missing signature instruction index validation in Ed25519 verification pattern	Low	Fixed
[L-03] Missing cancellation mechanism for pending transfers leads to operational inflexibility	Low	Fixed

7.1 High Severity Findings

[H-01] Missing caller authorization in initialize_vault allows initialization frontrunning leading to complete vault takeover

Affected files

solana-program/lib.rs lines 19-34
solana-program/lib.rs lines 268-292

Description

The initialize_vault function does not validate the caller's identity, allowing anyone to call it and become the vault authority. Since Solana programs require manual initialization (unlike Ethereum's constructor pattern), the first caller to successfully execute initialize_vault gains complete control over the vault.

The vulnerable code at lines 268-292 shows no constraint on the authority account:

```
#[derive(Accounts)]
pub struct InitializeVault<'info> {
    #[account(
        init,
        payer = authority,
        space = 8 + 32 + 32 + 32 + 1 + 1,
        seeds = [VAULT_SEED],
        bump
    )]
    pub vault: Account<'info, CasinoVault>,

    #[account(mut)]
    pub authority: Signer<'info>,  // No constraint on who can be authority

    pub system_program: Program<'info, System>,
}
```

At line 24, the caller becomes the vault authority without any verification:

```
vault.authority = ctx.accounts.authority.key();
```

Vulnerable Scenario:

The following steps help understand the issue:

1. Protocol team deploys the casino vault program to Solana mainnet.
2. Attacker monitors blockchain for new program deployments or observes the deployment transaction in mempool.

3. Attacker calls initialize_vault before the legitimate deployer, passing their own address as authority and an arbitrary withdrawal_signer.
4. The vault PDA is initialized with the attacker as authority at line 24.
5. Legitimate deployer's initialization transaction fails because the vault PDA already exists (Anchor's init constraint prevents re-initialization).
6. Attacker now has permanent control over vault operations including emergency_pause, resume, set_authority, and set_withdrawal_signer.

Impact

Complete vault takeover is possible if an attacker successfully frontruns the initialization. The attacker gains full administrative control including the ability to pause withdrawals, rotate the withdrawal signer key, and transfer authority. All future user deposits would be at risk under attacker control with no recovery mechanism available.

Recommendations

Add two new accounts to the InitializeVault struct with constraints that verify the caller is the program's upgrade authority:

```
#[derive(Accounts)]
pub struct InitializeVault<'info> {
    // ... existing accounts ...

    #[account(mut)]
    pub authority: Signer<'info>,

    // ADD THESE TWO ACCOUNTS:

    /// Verify caller is the program's upgrade authority
    #[account(
        constraint = program_data.upgrade_authority_address == Some(authority.key())
            @ ErrorCode::Unauthorized
    )]
    pub program_data: Account<'info, ProgramData>,
```

```
    /// CHECK: Verify program data account matches this program
    #[account(
        constraint = program.programdata_address()? == Some(program_data.key())
    )]
    pub program: Program<'info, CasinoVault>,

    pub system_program: Program<'info, System>,
}
```

The key additions are:

- **program_data account**: Holds the program's upgrade authority and enforces that authority.key() matches the upgrade authority
- **program account**: Links the program data to the current program to prevent using another program's upgrade authority. This ensures only the account that deployed the program can initialize the vault, preventing initialization frontrunning attacks.

Fair Casino:

Fixed

Zealynx:

Verified

7.2 Medium Severity Findings

[M-01] Missing token account constraints leads to potential mint confusion and unauthorized token transfers

Affected files

solana-program/lib.rs Lines 311-315

Description

The ProcessWithdrawal instruction context fails to validate critical properties of the token accounts used in the withdrawal operation. Specifically, vault_token_account and recipient_token_account lack constraints to verify:

1. **Vault token account ownership**: No validation that vault_token_account is owned by vault_authority.
2. **Mint matching:** No validation that either token account matches the expected vault.token_mint.

Current implementation:

```
#[account(mut)]
pub vault_token_account: Account<'info, TokenAccount>,

#[account(mut)]
pub recipient_token_account: Account<'info, TokenAccount>,
```

The only validation performed is a runtime check at line 151 that recipient_token_account.owner matches the authorized recipient, but this does not validate the mint or the vault account's ownership.

Impact

Attackers can potentially manipulate token account inputs to withdraw wrong tokens or bypass mint restrictions. While the SPL Token Program provides some protection by verifying authority signatures, the lack of mint and ownership validation creates opportunities for confusion attacks and could allow transfers of unintended token types. This is particularly critical given that vault.token_mint is stored but never enforced, making the entire mint tracking system ineffective.

Recommendations

Add explicit constraints to the vault_token_account and recipient_token_account in ProcessWithdrawal:

```
// vault_token_account: Add ownership and mint validation
#[account(
    mut,
    constraint = vault_token_account.owner == vault_authority.key() @ ErrorCode::InvalidVaultAccount,
    constraint = vault_token_account.mint == vault.token_mint @ ErrorCode::InvalidMint
)]
pub vault_token_account: Account<'info, TokenAccount>,
// recipient_token_account: Add mint validation
#[account(
    mut,
    constraint = recipient_token_account.mint == vault.token_mint @ ErrorCode::MintMismatch
)]
pub recipient_token_account: Account<'info, TokenAccount>,
```

Additionally, ensure vault.token_mint is properly validated during initialization:

```
#[account(
    constraint = token_mint.owner == &spl_token::ID @ ErrorCode::InvalidMint
)]
pub token_mint: Account<'info, Mint>,
```

Define the new error codes:

```
#[msg("Token account not owned by vault authority")]
InvalidVaultAccount,
#[msg("Token mint does not match vault mint")]
InvalidMint,
#[msg("Recipient token mint does not match vault mint")]
MintMismatch,
```

Fair Casino:

Fixed

Zealynx:

Verified

7.3 Low Severity Findings

[L-01] Single-step authority transfer without confirmation risks permanent loss of vault control on operator error

Affected Files

- solana-program/lib.rs#L215-L237
- solana-program/lib.rs#L241-L263

Description

The FAIR Casino vault implements authority transfer functions set_authority and set_withdrawal_signe that immediately transfer control in a single transaction without requiring confirmation from the new authority. This creates a critical risk where a single typographical error can result in permanent loss of vault control and all deposited funds.

Impact

If an administrator enters an incorrect address during authority transfer, vault control would be permanently lost with no recovery mechanism. This would prevent all administrative operations including emergency pause and future key rotation. The vault manages real user funds, and code comments reference planned "Turnkey migration," making authority transfers an anticipated operational requirement rather than a theoretical edge case.

Recommendations

Implement a two-step authority transfer pattern requiring explicit acceptance from the new authority.

1. **Update CasinoVault struct** (add 2 fields):

```
#[account]
pub struct CasinoVault {
    pub authority: Pubkey,
    pub withdrawal_signer: Pubkey,
    pub token_mint: Pubkey,
    pub vault_bump: u8,
    pub is_paused: bool,
    pub pending_authority: Option<Pubkey>,          // Add this
    pub pending_withdrawal_signer: Option<Pubkey>,  // Add this
}
```

Note: Increase space in InitializeVault from 106 to 172 bytes (adding 66 bytes for two Option fields).

2. Modify set_authority to nominate instead of transfer:

```
pub fn nominate_authority(ctx: Context<AdminControl>, new_authority: Pubkey) -> Result<()> {
    let vault = &mut ctx.accounts.vault;
    require!(ctx.accounts.authority.key() == vault.authority, ErrorCode::Unauthorized);
    require!(new_authority != Pubkey::default(), ErrorCode::InvalidAddress);

    vault.pending_authority = Some(new_authority);  // Store pending instead of transferring
    msg!("Authority nominated: {}. Must call accept_authority to complete", new_authority);
    Ok(())
}
```

3. Add acceptance function (new authority must call this):

```
pub fn accept_authority(ctx: Context<AcceptAuthority>) -> Result<()> {
    let vault = &mut ctx.accounts.vault;
    require!(Some(ctx.accounts.new_authority.key()) == vault.pending_authority, ErrorCode::Unauthorized);

    vault.authority = ctx.accounts.new_authority.key();
    vault.pending_authority = None;
    msg!("Authority transfer completed");
    Ok(())
}

#[derive(Accounts)]
pub struct AcceptAuthority<'info> {
    #[account(mut, seeds = [VAULT_SEED], bump)]
    pub vault: Account<'info, CasinoVault>,
    pub new_authority: Signer<'info>,
}
```

4. Apply same pattern to set_withdrawal_signer:

- Rename to nominate_withdrawal_signer and set pending_withdrawal_signer
- Add accept_withdrawal_signer function

Fair Casino:

Fixed

Zealynx:

Verified

[L-02] Missing signature instruction index validation in Ed25519 verification pattern

Affected Files

solana-program/lib.rs lines 60-145

Description

The Ed25519 signature verification implementation validates that public key and message data are read inline (pubkey_ix_index and msg_ix_index both equal 0xFFFF) but does not validate the signature_ix_index field. According to industry best practices for Ed25519 verification via instruction introspection, all three instruction index fields must be validated to ensure complete inline mode operation.

The Ed25519 instruction header structure contains three instruction index fields at bytes 4-5 (signature_ix_index), 8-9 (pubkey_ix_index), and 14-15 (msg_ix_index). When these fields equal 0xFFFF, it indicates the data should be read from the current Ed25519 instruction itself (inline mode) rather than from other instructions in the transaction.

Current implementation at lines 101-120:

```
// Read offsets and instruction indices from header
let pubkey_offset = u16::from_le_bytes([data[6], data[7]]) as usize;
let pubkey_ix_index = u16::from_le_bytes([data[8], data[9]]);

let msg_offset = u16::from_le_bytes([data[10], data[11]]) as usize;
let msg_size = u16::from_le_bytes([data[12], data[13]]) as usize;
let msg_ix_index = u16::from_le_bytes([data[14], data[15]]);

// CRITICAL SECURITY CHECK (Prevents Offset Attack):
const CURRENT_IX_INDEX_SENTINEL: u16 = u16::MAX;
require!(
    pubkey_ix_index == CURRENT_IX_INDEX_SENTINEL,
    ErrorCode::InvalidEd25519InstructionData
);
require!(
    msg_ix_index == CURRENT_IX_INDEX_SENTINEL,
    ErrorCode::InvalidEd25519InstructionData
);
```

The signature_ix_index field at bytes 4-5 is never read or validated. The validation gap represents an incomplete implementation of the recommended Ed25519 verification security pattern. It is recommended to validate all three instruction index fields to ensure the verification instruction operates in a fully constrained inline mode.

Recommendations:

Add the signature instruction index validation after line 103. Read the signature_ix_index field and validate it matches the CURRENT_IX_INDEX_SENTINEL value:

```
let sig_ix_index = u16::from_le_bytes([data[4], data[5]]);
require!(
    sig_ix_index == CURRENT_IX_INDEX_SENTINEL,
    ErrorCode::InvalidEd25519InstructionData
);
```

This completes the inline mode validation pattern by ensuring the signature, public key, and message are all read from the same Ed25519 instruction.

Fair Casino:

Fixed

Zealynx:

Verified

[L-03] Missing cancellation mechanism for pending transfers leads to operational inflexibility

Affected Files

solana-program/lib.rs

Description

The two-step transfer pattern for authority and withdrawal_signer lacks a cancellation mechanism. Once a nomination is initiated, the current authority cannot revoke it - they can only overwrite it with a different nomination. This creates operational inflexibility if a nomination was made in error.

Recommendations

Add cancellation instructions:

```
pub fn cancel_pending_authority(ctx: Context<CancelPending>) -> Result<()> {
    let transfers = &mut ctx.accounts.pending_transfers;
    transfers.pending_authority = Pubkey::default();
    msg!("Pending authority transfer cancelled");
    Ok(())
}
```

Fair Casino:

Fixed

Zealynx:

Verified