ZEALYNX

Your Web3 Security partner

Fair Casino

TypeScript Audit

January 27, 2026

Prepared by Zealynx
contact@zealynx.io

Fernando @0xMrjory

Contents

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

This penetration testing assessment was conducted within the defined scope and timeframe agreed upon with the client. While every effort was made to identify security vulnerabilities, this assessment cannot guarantee the discovery of all weaknesses or ensure complete security of the tested systems. The findings represent a point-in-time evaluation; subsequent changes to systems, configurations, or the threat landscape may introduce new vulnerabilities. The assessor assumes no liability for security incidents occurring after the assessment or for vulnerabilities outside the agreed scope. This report is confidential and intended solely for the authorized recipient. Testing was performed with proper authorization and in compliance with applicable laws and regulations.

3. Overview

3.1 Project Summary

A security audit of the Fair Casino TypeScript backend services and frontend verification utilities was conducted by Zealynx Security with a focus on security assessment and risk identification. We performed the security evaluation based on the agreed scope during 4 days.

Following our systematic approach and methodologies, testing was performed on critical money flows including deposits, withdrawals, and provably-fair game integrity. Our security assessment revealed 2 issues, comprising 2 High severity findings.

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 follow-on audit phase covering the SOL→FAIR swap flow, comprising ~1,500 nSLOC across 7 files. The backend implements Jupiter DEX integration for swap quote retrieval and unsigned transaction building (/api/fair/quote, /api/fair/swap), with price feed services for real-time token pricing. The frontend provides a client-side fail-closed transaction allowlist (fairSwapSafety) that validates swap transactions before Phantom wallet signing, ensuring only whitelisted instruction patterns are permitted.

4. Audit Methodology

Approach

During our security assessments, we uphold a rigorous approach to maintain **high-quality standards. Our methodology encompasses thorough TypeScript Code Review, Web Application Security Testing**, **Infrastructure Assessment**, and meticulous manual **penetration testing**.

Throughout the TypeScript application audit and penetration testing process, we prioritize the following aspects to uphold excellence:

1. **Front-End Security Review**: We comprehensively evaluate client-side code quality, security implementations, and potential attack vectors including XSS, CSRF, and clickjacking vulnerabilities.
2. **Back-End Architecture Analysis**: Our assessments emphasize secure coding practices, authentication mechanisms, authorization controls, session management, and data validation to ensure robust server-side security.
3. **Infrastructure and Configuration**: We meticulously review deployment configurations, environment variables, database connections, and third-party integrations to identify misconfigurations and security gaps.
4. **Penetration Testing**: We conduct both automated and manual testing against OWASP Top 10 vulnerabilities and custom threat scenarios, including real-world exploitation of identified weaknesses to demonstrate actual risk impact.

Testing Execution

During the Fair Casino engagement, Zealynx performed the following actions to ensure thorough coverage of the defined scope:

• Confirmed the precise scope of backend services, frontend utilities, and testing windows in coordination with Fair Casino.
• Mapped the money flow attack surface and reviewed application logic to identify critical assets including deposits, withdrawals, and game settlement.
• Conducted automated vulnerability scanning on backend endpoints, targeting OWASP Top 10 risks and common web vulnerabilities.
• Performed manual security testing to simulate real-world attacker behavior, including input fuzzing, WebSocket manipulation, authentication bypass attempts, and race condition testing.

• Reviewed provably-fair implementation for cryptographic correctness, commit-reveal integrity, and seed entropy guarantees.
• Assessed API security focusing on idempotency, rate-limiting, and safe failure modes.

Validation Process

After initial testing and reporting, Zealynx followed a structured process to ensure the accuracy and effectiveness of remediation:

- Documented all findings with supporting evidence, impact analysis, and actionable remediation guidance.
- Provided detailed reports to Fair Casino and communicated directly to clarify technical details and remediation steps.
- For each issue marked as "fixed" by Fair Casino, re-tested the affected components to confirm successful mitigation and closure.
- Updated the final report to reflect the status of each finding, distinguishing between issues confirmed as resolved and those pending further action.

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

Over a 4-day engagement, the Zealynx Security team conducted a security audit of the Fair Casino TypeScript backend services and frontend verification utilities. The assessment was conducted from January 20-24, 2026.

The audit focused on identifying vulnerabilities, logic flaws, and implementation-level issues affecting the Fair Casino platform's critical money flows, including deposits, withdrawals, and provably-fair game integrity.

A total of 2 issues were identified and categorized as follows:

- 2 High severity

The initial commit hash audited is: 61b0800777e7e0b1574f783d7b87925e26fb45b8

6.1 The Key Findings

• **Frontend Trusts WebSocket Balance Events as Authoritative, Enabling UI Spoofing:** The frontend accepts balance_update WebSocket events and updates the displayed balance without validating message authenticity, session binding, or server truth. An attacker intercepting WebSocket traffic can modify the newBalance value, causing the UI to display fabricated balances and enabling state desynchronization.

• **WebSocket Deposit Events Can Be Spoofed, Causing False Confirmations:** The frontend blindly trusts deposit_confirmed WebSocket events and immediately updates UI state without verifying the message originated from the server or that the deposit corresponds to an on-chain transaction. Injecting a crafted event with an arbitrary amount triggers false deposit notifications and balance updates, causing severe frontend/backend desynchronization.

6.2 Architectural Security Observations

During the security assessment, we observed several architectural decisions and controls that contribute to Fair Casino's security posture:

- **Two-Key Authorization Model:** Separation between vault authority and withdrawal signer ensures no single key can authorize fund movements. This limits the blast radius of key compromise.
- **Secure Enclave Key Custody:** Private keys are managed through Turnkey and never reside in backend memory, mitigating hot wallet theft risks.
- **On-Chain Signature Verification:** Ed25519 withdrawal authorization is enforced at the Solana program level, preventing bypass through frontend or API tampering.
- **Circuit Breaker Controls:** Vault health monitoring enables automated pause on suspicious outflows, limiting potential catastrophic loss.
- **Idempotent Balance Operations:** All money movements are keyed and logged, ensuring safe retries and crash resilience across the deposit and withdrawal flows.
- **Fail-Closed Behavior:** Operations are denied on uncertainty, prioritizing safety over availability.

6.3 Security Strengths Observed

Based on our assessment, Fair Casino demonstrates the following security strengths:

- **Cryptographic Authorization:** The combination of commit-reveal schemes, HMAC-SHA256 RNG, and Ed25519 signature verification provides robust, multi-layered protection against outcome manipulation and unauthorized withdrawals.
- **Dual RPC Verification:** Independent RPC confirmation before crediting deposits protects against spoofed or inconsistent blockchain data.
- **Distributed Locking & Fencing:** Prevents concurrent withdrawal race conditions, ensuring transactional safety across parallel requests.
- **Deterministic Auditability:** Nonce-based replay and server-side game state storage enable full game reproducibility for audit and dispute resolution.
- **Defensive-by-Default Design:** The platform rejects unsafe inputs predictably and avoids processing malformed requests, supporting reliability and resilience.
- **No Critical Compromise Paths Identified:** The majority of findings were focused on fairness assurance mechanics and UX consistency rather than direct fund-loss paths.

Summary of Findings:

Vulnerability	Severity	Status
[H-01] Frontend trusts WebSocket balance_update events as authoritative, enabling UI balance spoofing and state desync	High	Fixed
[H-02] WebSocket deposit_confirmed events can be spoofed client-side, causing false deposit confirmations	High	Fixed

7. Audit Findings

7.1 High Severity Findings

[H-01] Frontend trusts WebSocket balance_update events as authoritative, enabling UI balance spoofing and state desync

Affected files

WebSocket message validation

Description

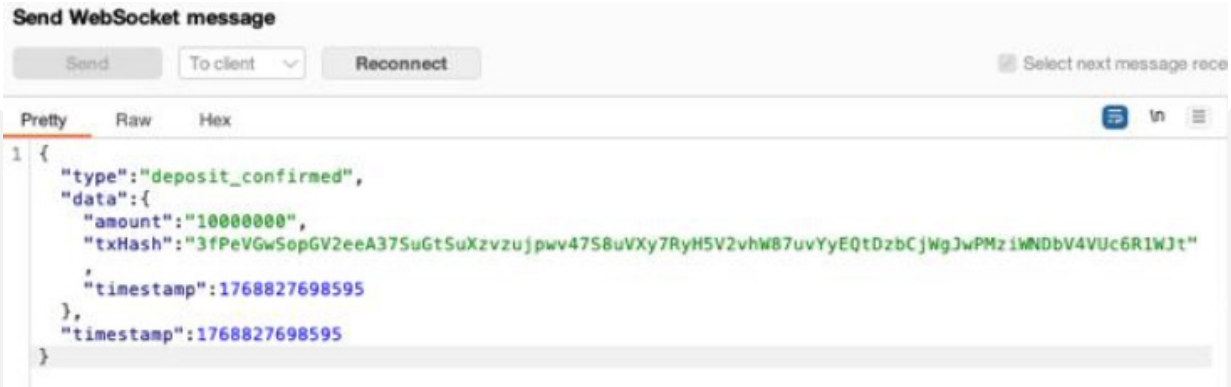
The frontend appears to accept balance_update WebSocket events and update the user's displayed balance without validating that:

- the message is authentic (origin integrity)
- the update corresponds to the current authenticated user session
- the new balance matches server truth (e.g., via a confirmatory API fetch)

As a result, a user (or attacker with the ability to tamper with WebSocket traffic) can spoof the UI balance and put the client into an inconsistent state.

Vulnerable Scenario:

1. Authenticate and open the app.
2. Intercept a legitimate balance_update message in the WebSocket stream.
3. Modify newBalance value (e.g., from 4.238334 to 5000000.238334).
4. UI updates immediately, showing an incorrect available balance.

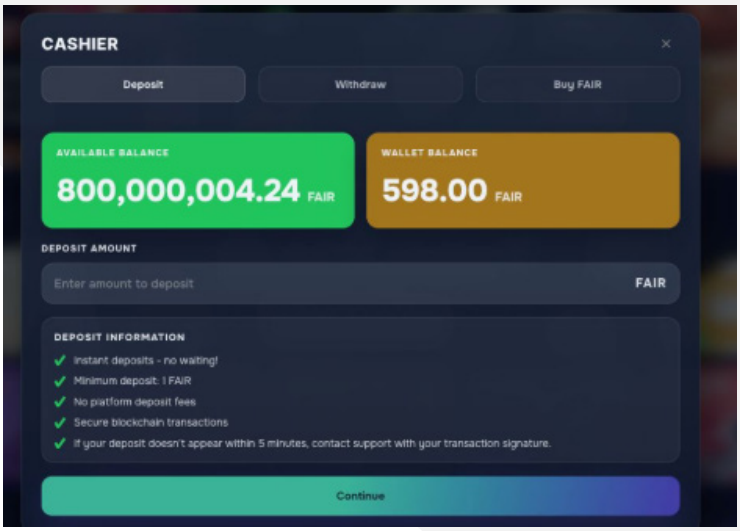


Message	Direction	Manual	Length	Time
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	202	14:05:08 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	202	14:05:09 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	202	14:05:09 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:52 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:54 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	205	14:05:54 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	217	14:06:52 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	217	14:06:53 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	217	14:07:03 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	220	14:07:07 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	224	14:07:20 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	224	14:11:23 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	224	14:11:34 19 Jan 202
{"type":"balance_update","data":{"walletAdd...	← To client	✓	224	14:11:41 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:12:51 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:12:53 19 Jan 202

{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:12:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:12:53 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:12:54 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:12:54 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:54 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:55 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:55 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:55 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:55 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:55 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:56 19 Jan 202
{"type":"deposit_confirmed","data":{"amount...	← To client	✓	209	14:13:57 19 Jan 202

Pretty Raw Hex


```
{
  "type":"deposit_confirmed",
  "data":{
    "amount":"10000000",
    "txHash":"3fPeVGwSopGV2eeA37SuGtSuXzvzujpwv47S8uVXy7RyH5V2vhW87uvYyEQtDzbCjWgJwt
    ,
    "timestamp":1768827698595
  },
  "timestamp":1768827698595
}
```

Impact

- **UI integrity loss:** user interface can show a fabricated balance.
- **State desync:** UI may enable/disable actions based on false balance, confusing users and testers.

Recommendations

- Treat WebSocket messages as advisory, not authoritative:
- On balance_update, trigger a refresh from GET /api/user/balance (or equivalent) and only update UI using API truth.
- Validate message binding:
- Ensure the walletAddress matches the authenticated wallet in the client context.
- Consider including a server-side sessionId / userId and validate it client-side.

Fair Casino:

Fixed

Zealynx:

Verified

[H-02] WebSocket deposit_confirmed events can be spoofed client-side, causing false deposit confirmations

Affected files

Websocket connection

Description

The frontend WebSocket client blindly trusts incoming server-side events such as deposit_confirmed and immediately updates UI state (balance, deposit status, notifications) without verifying:

- that the message originated from the trusted server
- that the deposit corresponds to an on-chain transaction

A fabricated deposit_confirmed event with an arbitrary amount is accepted:

- The UI reflects the fake deposit and balance update
- No immediate backend reconciliation corrects the state

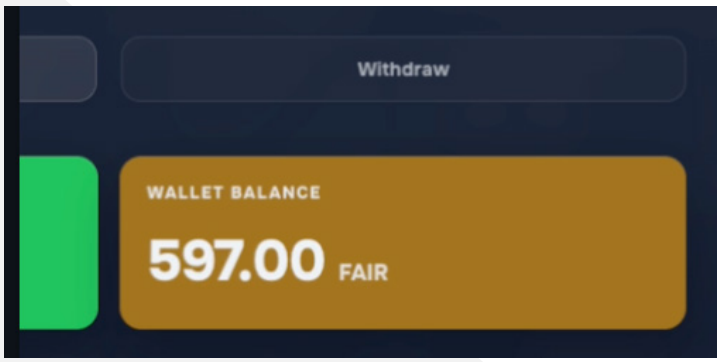
This creates false-positive deposits in the UI, causing severe frontend/backend desynchronization.

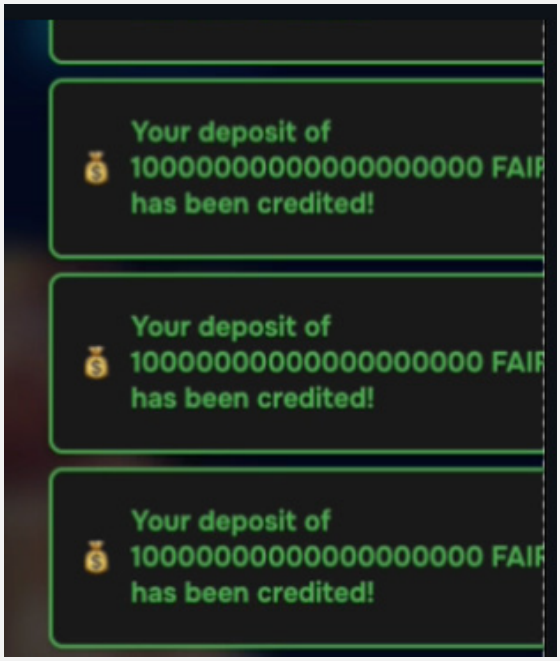
Vulnerable Scenario;

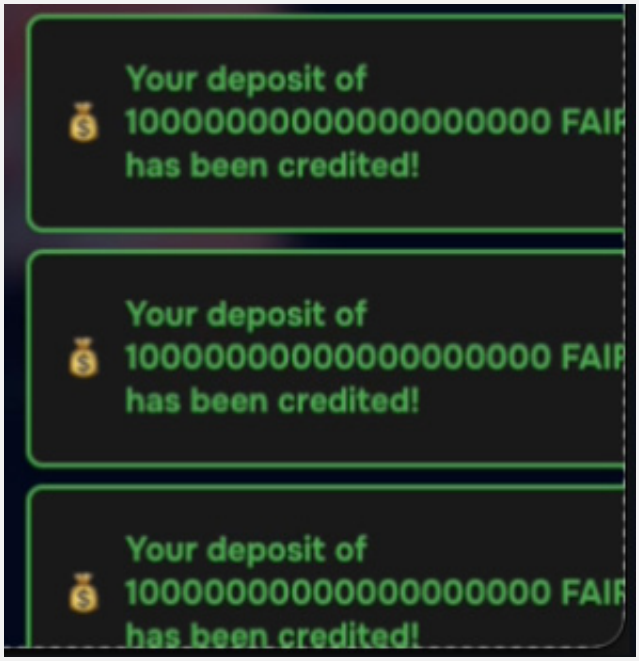
1. Authenticate normally.
2. Open WebSocket connection to /ws.
3. Intercept or inject a crafted deposit_confirmed message with a large amount.


```
{
  "type": "deposit_confirmed",
  "data": {
    "amount": "1000000000000000",
    "txHash": "2yNuJFuRHDDS4V03Ahb6GizcPi5mPpeYZHZZiWNQnTvFJt5CkPALWAWJZAYersE9wFmVPPHMDHtsbTwV9dNuq",
    "timestamp": 1768907210437
  },
  "timestamp": 1768907210437
}
```


5. Observe deposit notification shown







Impact

- **False deposit confirmations:** Users can see deposits that never occurred.
- **Severe UI integrity failure:** Frontend state diverges from blockchain and backend truth.

Recommendations

- Make WebSocket events advisory, not authoritative
- On deposit_confirmed, trigger a fetch to:
  GET /api/user/balance
  GET /api/deposits/{txHash}
- Update UI only after API confirms backend truth.

Fair Casino:

Fixed

Zealynx:

Verified