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 7 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 13 issues, comprising 4 High, 6 Medium, and 3 Low 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 TypeScript backend and frontend codebase comprising ~6,500 nSLOC. The backend implements critical money flows including atomic balance mutations with idempotency guarantees, provably-fair game outcome derivation using commit-reveal cryptography, deposit monitoring with SPL token transfer validation, unified withdrawal processing with challenge-signature authorization, and circuit-breaker controls via vault health monitoring. The frontend provides client-side provably-fair verification utilities.

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 14-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 5-19, 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 13 issues were identified and categorized as follows:

- 4 High severity
- 6 Medium severity
- 3 Low severity

The initial commit hash audited is: 955864f390f5599bd2fcbc3702fe460dd354efed

6.1 The Key Findings

• **Wallet Authentication Blocked Due to Unstable Challenge Endpoint:** The authentication endpoint exhibits unstable behavior returning HTTP 429 and 500 errors, effectively blocking wallet login and resulting in complete denial of service for legitimate users.
• **Deposit Credited to Wrong User Due to Sender Inference Fallback:** When parsing deposits, the fallback logic uses the first transaction signer as the sender, which in complex transactions may not represent the actual depositor, resulting in funds credited to the wrong account.
• **Server Can Control Both Seeds When Client Seed Is Omitted:** The provably fair system allows the server to generate both serverSeed and clientSeed when the client seed is not provided, enabling a biased operator to cherry-pick seed pairs that produce house-favorable outcomes while still passing verification.
• **Server Can Choose serverSeed After Seeing clientSeed:** The session initialization allows the server to observe the clientSeed before committing to serverSeedHash, breaking the commit-reveal order and enabling outcome manipulation without detection.

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] Wallet Authentication Blocked Due to Unstable Challenge Endpoint (429 / 500), Resulting in Persistent Login Failure	High	Fixed
[H-02] Deposit credited to wrong user due to sender inference fallback	High	Fixed
[H-03] Design Finding — Server Can Control Both Seeds When Client Seed Is Omitted (Weakens “Provably Fair” Guarantees)	High	Fixed
[H-04] Design Finding — Server Can Choose serverSeed After Seeing clientSeed (Seed-Cherry-Picking Risk)	High	Fixed
[M-01] Client seed accepts whitespace/low-entropy values and has no length/charset bounds (potential DoS + undermines “user influence”)	Medium	Fixed
[M-02] WebSocket Crash Events Broadcast to All Clients Without Subscription Enforcement (Privacy Leak + Amplification/DoS Risk)	Medium	Fixed
[M-03] WebSocket JWT validation bypasses centralized HTTP security controls	Medium	Fixed
[M-04] Verification Logic Uses Inconsistent Hash/Derivation vs Gameplay	Medium	Fixed
[M-05] Game bet amount accepts hexadecimal strings (numeric coercion bypass)	Medium	Fixed

Summary of Findings :

Vulnerability	Severity	Status
[M-06] Duplicate start Mines request desynchronizes client/server state, freezing Mines UI until re-auth/refresh	Medium	Fixed
[L-01] Fairness “Verifier” can generate “Verification Successful” results for games that never occurred (not bound to game history)	Low	Fixed
[L-02] Insecure Randomness in Session ID Generation	Low	Fixed
[L-03] Reflected Request Path in Error Response (Unnormalized Route Handling)	Low	Fixed

7. Audit Findings

7.1 High Severity Findings

[H-01] Wallet Authentication Blocked Due to Unstable Challenge Endpoint (429 / 500), Resulting in Persistent Login Failure

Affected files

Wallet authentication (Phantom → backend challenge flow)
/api/auth/challenge?walletAddress=<walletAddr>

Description

The dApp's wallet authentication mechanism relies on a challenge–response flow initiated via the backend endpoint:
GET /api/auth/challenge?walletAddress=<wallet>

Based on direct observation using browser developer tools, this endpoint exhibits unstable behavior, returning both:

- HTTP 429 – AUTH_CHALLENGE_RATE_LIMITED
- HTTP 500 – Internal Server Error

When this occurs, the authentication flow fails before Phantom is able to request a signature, effectively blocking wallet login entirely.

From the user's perspective, this manifests as:

- Phantom displaying warning or error states
- Authentication never completing
- The dApp appearing "blocked" or unusable

Because wallet authentication is a hard dependency for accessing the application, this issue results in a complete denial of service for legitimate users.

Vulnerable Scenario:

1. Visit the dApp.
2. Attempt authentication challenges via the phantom wallet.
3. Observe request to:

GET /api/auth/challenge?walletAddress=<wallet>

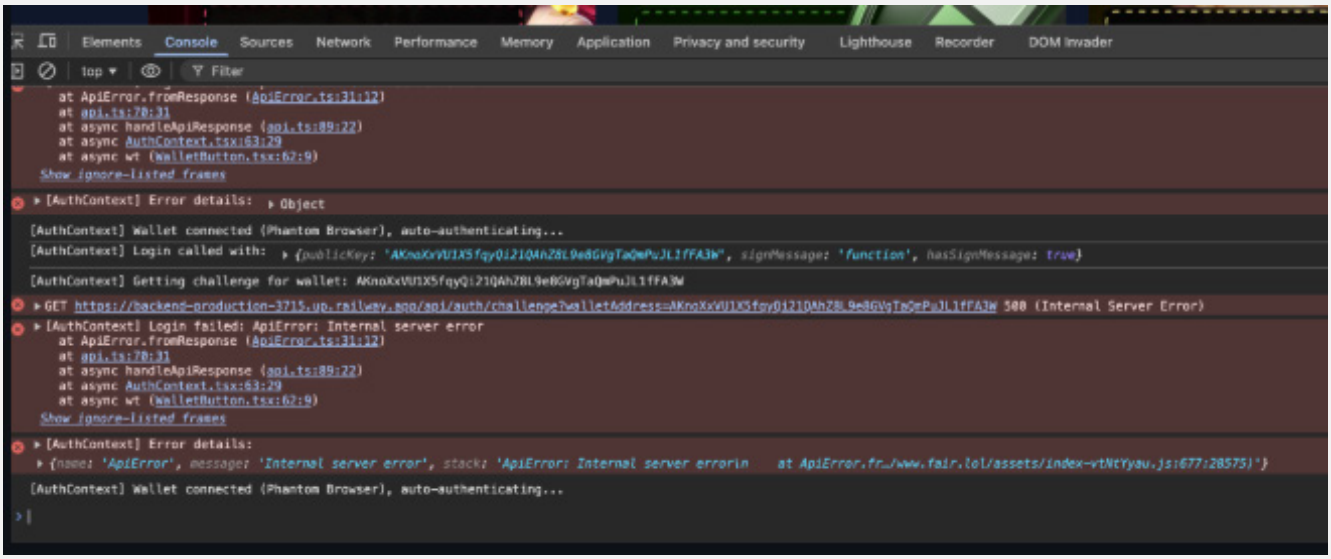
After several attempts:

Receive *429 AUTH_CHALLENGE_RATE_LIMITED*, or
Receive *500 Internal Server Error*.

5. Authentication never proceeds to Phantom signature request.

Rate-limiting response (HTTP 429)
The backend responds with a structured rate-limit error:


```
{
  "success": false,
  "error": "AUTH_CHALLENGE_RATE_LIMITED",
  "message": "Too many challenge requests. Please try again in a minute.",
  "retryAfter": 43,
  "limit": 20,
  "window": "60 seconds"
}
```

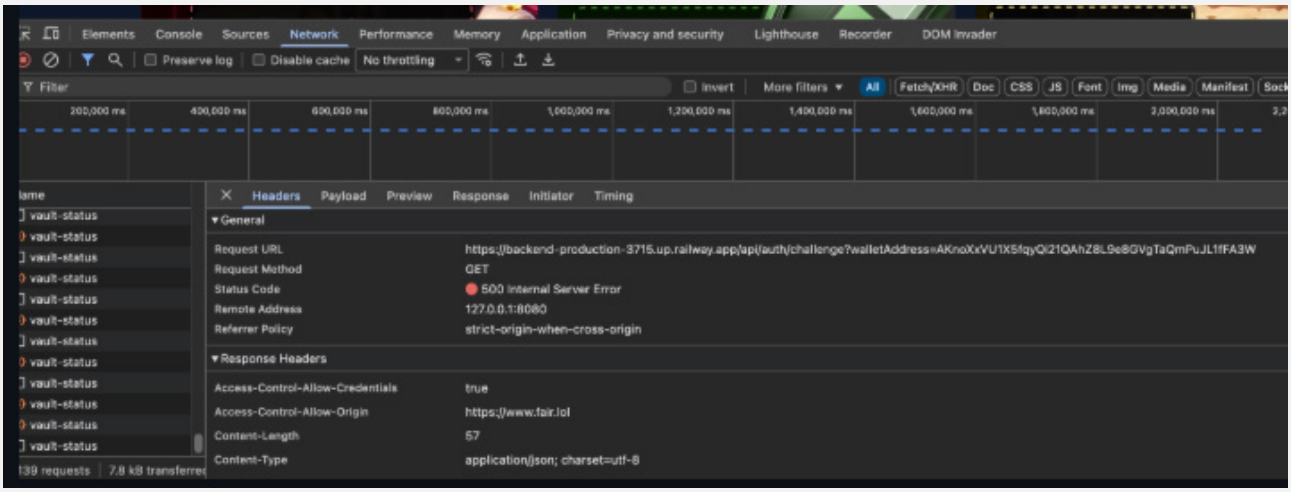



Notably, additional rate-limit headers are also present (X-Ratelimit-*), indicating multiple rate-limiting layers (likely application + edge/CDN), which may not be synchronized.

Internal server error (HTTP 500)

In subsequent attempts, the same endpoint returns:

HTTP/2 500 Internal Server Error



This occurs on the same request path (/api/auth/challenge) and prevents authentication even after waiting for the advertised rate-limit window to expire.

The following video ilustrates the error codes receieved during the interaction with the backend endpoint: *https://github.com/ZealynxSecurity/Internal-FAIR-Labs-Audit/raw/main/.github/ISSUE_TEMPLATE/auth_issue.mov*

Impact

- Complete authentication failure: Users cannot log in or interact with the dApp.
- Effective denial of service: The application becomes unusable.
- User trust degradation: From the user's perspective, the failure appears as a Phantom wallet block or "malicious dApp" warning.
- Potential targeted lockout: If the limiter is keyed by wallet address, an attacker could intentionally trigger rate limits to block specific wallets.

Recommendations

Backend:

- Make the challenge endpoint idempotent.
- Reuse an existing valid challenge per wallet within its TTL.
- Align rate-limit enforcement with actual reset behavior.
- Avoid wallet-only rate-limit keys (use IP + wallet).
- Handle backend dependency failures gracefully (return controlled 503, not 500).
- Add structured logging to identify limiter keys and failure causes.

Frontend:

- Ensure challenge requests are issued once per user action.
- Disable auto-retry loops.
- Respect Retry-After and block further attempts until expiry.

Infrastructure (if applicable):

- Audit edge/CDN rate limiting to ensure it does not conflict with application-level limits. Ensure rate-limit headers accurately reflect actual enforcement.

Fair Casino:

Fixed

Zealynx:

Verified

[H-02] Deposit credited to wrong user due to sender inference fallback

Affected files

backend/src/services/deposit-monitor.ts

Description

When parsing deposits, the system tries to determine the sender wallet from parsed token program instructions. If that produces *unknown*, it falls back to using the first signer in the transaction message as the "sender." In complex transactions (multi-instruction, aggregator involvement, multisig, relayers), "first signer" may not represent the depositor. This can result in a deposit being credited to the wrong casino account.

Vulnerable Scenario;

1. Create a transaction that results in the vault token account's FAIR balance increasing (valid deposit), but with multiple signers/instructions such that:
    - the parsed transfer info returns source/authority as a token account or something that hits your fallback logic, and
    - the "first signer" in accountKeys is not the actual depositor.

2. Submit it and observe the credited from wallet address differs from the actual depositor wallet.

Fair Casino:

Fixed

Zealynx:

Verified

[H-03] Design Finding — Server Can Control Both Seeds When Client Seed Is Omitted (Weakens “Provably Fair” Guarantees)

Affected files

backend/src/domain/provably-fair.ts

Description

In provably fair system, the server must commit to its secret seed (by sending the hash) before receiving the client's seed to prevent the server from choosing a biased serverSeed based on the client's input. Here, startSession accepts an optional clientSeed parameter, generates the serverSeed after seeing it, and only then computes the hash. This reverses the proper order, allowing the server to iterate over potential serverSeeds until a favorable outcome (for the house) is achieved for known future nonces. Similarly, in rotateSession, a newClientSeed can be provided, with the same post-visibility generation of serverSeed.

Vulnerable Scenario;

1. Client starts a session without providing a *clientSeed*.
2. Server generates both *serverSeed* and *clientSeed*.
3. A biased server repeats generation + simulation offline:
    - A biased server repeats generation + simulation offline:
    - Choose a pair that produces desired outcomes (e.g., unfavorable Crash distribution).

4. Server commits to *serverSeedHash*, proceeds normally, and later reveals seeds.
5. Verifier confirms results as “valid” because they are internally consistent, even though the seed pair was cherry-picked.

Impact

- **Unfair games:** A rogue server operator can bias outcomes (e.g., force low dice rolls or early crashes) while maintaining "provable" hashes post-game, eroding trust and enabling house edges beyond advertised.
- **Regulatory/compliance risks**: Undermines the "provably fair" claim, potentially leading to user disputes, legal issues, or platform bans in jurisdictions requiring verifiable RNG.

Recommendations

- Preferred (strongest): enforce independent client input.

- Make *clientSeed* mandatory for *startSession* and *rotateSession*.
- If missing or whitespace-only, reject with a clear error and guidance.

If you keep a fallback, make it explicit and auditable:

- Label it clearly in UI: “Server-generated client seed (no user influence).”
- Require user acknowledgement before proceeding.

Fair Casino:
Fixed

Zealynx:
Verified

[H-04] Design Finding — Server Can Choose serverSeed After Seeing clientSeed (Seed-Cherry-Picking Risk)

Affected files

backend/src/domain/provably-fair.ts

Description

The session initialization API allows the client to provide clientSeedduring session creation/rotation. Because the backend receives (and therefore can observe) the clientSeed before it returns the serverSeedHash commitment to the client, a malicious operator could generate multiple candidate serverSeed values and select one that yields house-favorable outcomes for the known clientSeed across upcoming nonces. The server can then return the corresponding serverSeedHash. Later, when serverSeed is revealed, all results still “verify” because they are internally consistent, but the seed was cherry-picked.
This breaks a key provably-fair guarantee: the server must commit to its seed before it knows the client seed, otherwise the server can bias the outcome stream while preserving verifiability.

Strictly speaking:

- The server does generate serverSeed and serverSeedHash before it sets finalClientSeed.

But it still receives clientSeed as an input parameter, meaning the server knows it at function entry, so from a threat-model perspective, the server has visibility of clientSeed before it commits (because the “commit” is internal at that moment; it hasn’t been returned to the client yet).

Impact

- Outcome bias without detection: A rogue operator can manipulate expected value beyond the advertised house edge while still passing verification.
- Regulatory/compliance risk: Weakens/invalidates “provably fair” claims under an adversarial operator model.

Recommendations

- Refactor session creation: Generate serverSeed and send serverSeedHash first (e.g., via a commitServerSeed method). Then, have a separate setClientSeed method where the client provides their seed after the commitment.

- Remove the optional clientSeed from startSession and rotateSession; force clients to set it post-commitment. If clientSeed is server-generated (fallback), ensure it's non-malleable and auditable (e.g., derive from blockchain data or external oracle).

Fair Casino:
Fixed

Zealynx:
Verified

7.2 Medium Severity Findings

[M-01] Client seed accepts whitespace/low-entropy values and has no length/charset bounds (potential DoS + undermines “user influence”)

Affected files

frontend/src/components/provably-fair/FairnessCenterModal.tsx:378-385

```
<input
  type="text"
  ...
  value={newClientSeed}
  onChange={(e) => setNewClientSeed(e.target.value)}
/>
```

frontend/src/services/game.service.ts:113-115

Sends it directly:

```
api.post('/api/game/session/rotate', { clientSeed: newClientSeed })
```

Backend (no validation at controller):

backend/src/controllers/GameController.ts:529-535

```
const { clientSeed } = req.body;
const result = await this.gameService.rotateSession(authReq.user!.userId, clientSeed);
```

Backend (seed accepted as-is; whitespace passes):

backend/src/domain/provably-fair.ts:322-353

```
let clientSeedToUse = newClientSeed;

if (activeSession) {
  if (!clientSeedToUse) clientSeedToUse = activeSession.clientSeed;
}

// Default client seed if still not set
if (!clientSeedToUse) {
  clientSeedToUse = random.randomHex(16);
}
```

Description

The Fairness Center allows users to set an arbitrary clientSeed. The backend accepts clientSeed with no trimming, no charset validation, and no length limit.

As a result:

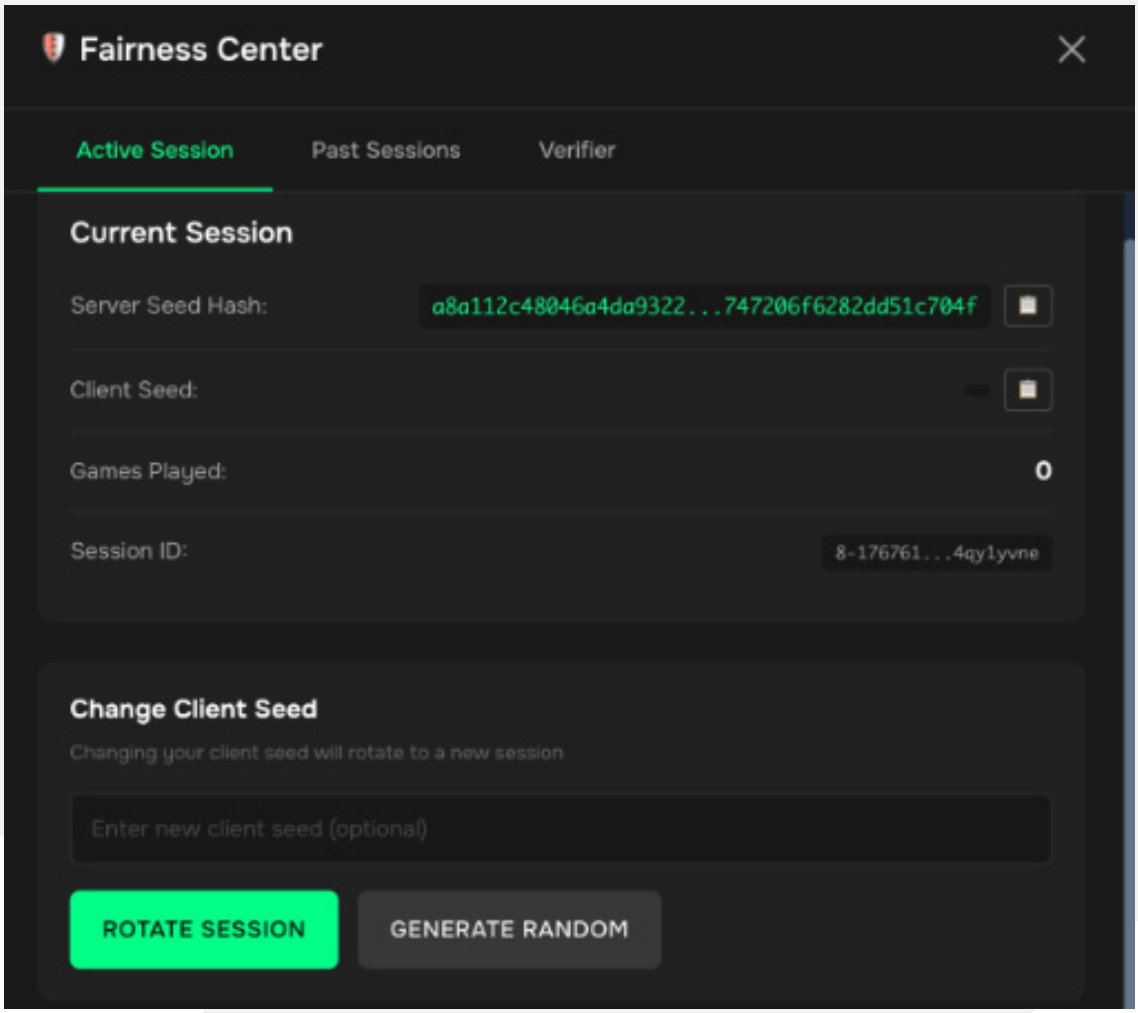
A user can set " " (space), "", or any tiny string → effectively no meaningful entropy from the client side.
A malicious client can send an extremely large string as clientSeed → the server will:

- store it in session state/history
- feed it into HMAC computations for every game outcome
- potentially increase payload sizes in APIs / WS events / logs
- This becomes a resource exhaustion primitive (CPU + memory + DB/storage growth), and can also destabilize verification tooling/UI.

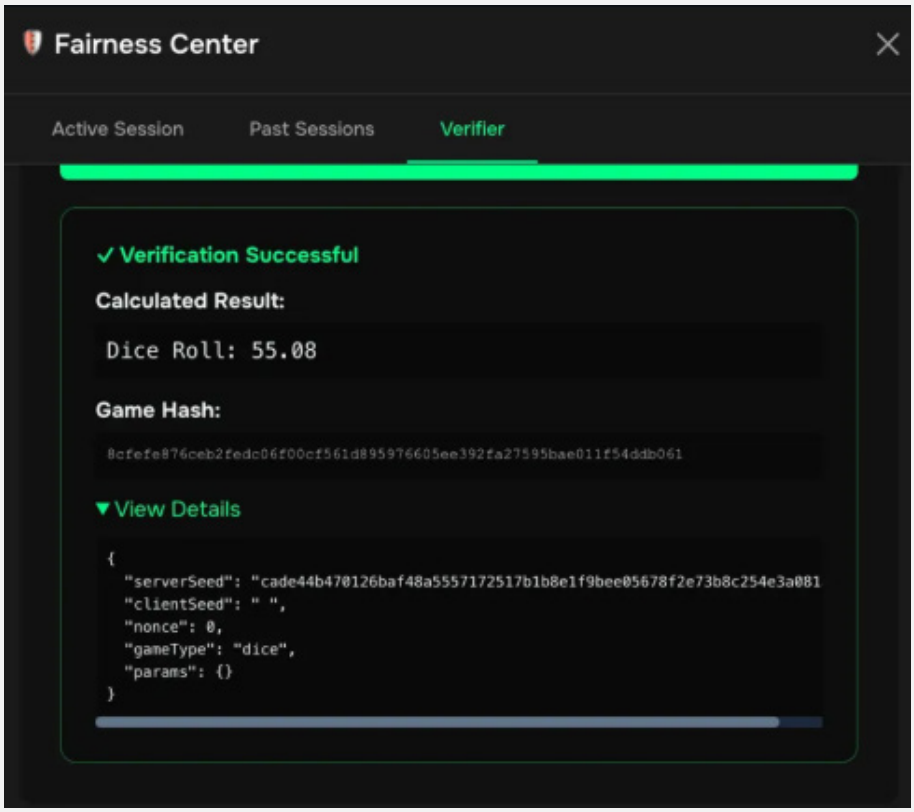
Vulnerable Scenario:
The following steps help understand the issue:

1. Low-entropy acceptance
    - In Fairness Center → set client seed to a single space " " → rotate session.

Observe backend accepts it and displays it as the active session clientSeed



and the verifier displays the value being effectively empty:



Impact

- Security / availability: attackers can degrade performance by using huge seeds (HMAC input bloat, DB bloat, larger responses, more GC pressure).
- Fairness assurance perception: “user influence” becomes misleading if users accidentally set trivial/blank seeds and believe it improves fairness.

Recommendations

Backend must enforce validation (frontend checks are not enough):

- Trim and normalize
- Treat whitespace-only seeds as empty
- seed = seed?.trim()
- if empty → generate random
- Enforce something like: minLen = 8 (or 16), maxLen = 128 (or 256).
- Reject anything above max with 400 Bad Request.
- Allow [a-zA-Z0-9_-] or hex only, to keep it stable across systems.
- In UI, show a warning if the seed is too short / low-entropy.

Fair Casino:

Fixed

Zealynx:

Verified

[M-02] Deposit credited to wrong user due to sender inference fallback

Affected files

backend/src/services/deposit-monitor.ts

Description

When parsing deposits, the system tries to determine the sender wallet from parsed token program instructions. If that produces unknown, it falls back to using the first signer in the transaction message as the "sender." In complex transactions (multi-instruction, aggregator involvement, multisig, relayers), "first signer" may not represent the depositor. This can result in a deposit being credited to the wrong casino account.

Vulnerable Scenario:

1. Create a transaction that results in the vault token account's FAIR balance increasing (valid deposit), but with multiple signers/instructions such that:
    - the parsed transfer info returns source/authority as a token account or something that hits your fallback logic, and
    - the "first signer" in accountKeys is not the actual depositor.

2. Submit it and observe the credited from wallet address differs from the actual depositor wallet.

Fair Casino:

Fixed

Zealynx:

Verified

[M-03] WebSocket JWT validation bypasses centralized HTTP security controls

Affected files

backend/src/websocket/WebSocketServer.ts

Description

The WebSocket authentication flow performs raw JWT verification using the low-level jsonwebtoken.verify() function directly with only the shared secret. This completely bypasses the centralized AuthManager.verifyToken() method used across all HTTP routes.The AuthManager.verifyToken() provides critical additional security controls, including:

- Token blacklist/revocation checks
- Enforcement of issuer and audience claims
- Standardized validation logic and error handling

By contrast, the WebSocket handler only validates the signature and standard timing claims, skipping all custom and revocation-related checks.

Vulnerable Scenario:

The following steps help understand the issue:

1. Authenticate via HTTP and obtain a valid JWT.
2. Establish a successful WebSocket connection using the token.
3. Trigger token revocation (e.g., logout via HTTP endpoint, which adds the token/JTI to the blacklist).
4. Attempt a new protected HTTP request → rejected (blacklist enforced by AuthManager).
5. Attempt a new WebSocket connection with the same revoked token → accepted (no blacklist check).
6. Existing WebSocket connection continues receiving real-time updates.

Impact

- Revoked tokens remain usable on WebSocket.
- Expanded attack surface from leaked tokens.
- WebSocket endpoint for persistent real-time monitoring or abuse.

Recommendations

- Reuse the centralized AuthManager.verifyToken() in the WebSocket handler:typescript

```
const decoded = await AuthManager.verifyToken(token);
```

- If direct reuse is not feasible, extract shared validation logic into a common utility function and invoke it from both HTTP and WebSocket paths.
- On successful revocation/logout events, proactively close any active WebSocket connections for the affected user (track by walletAddress or userId).

Fair Casino:

Fixed

Zealynx:

Verified

[M-04] Verification Logic Uses Inconsistent Hash/Derivation vs Gameplay

Affected files

backend/src/domain/provably-fair.ts

Description

The verifyResult method uses the private generateHash (simple SHA-256 concatenation) to compute the hash, but actual games use the public generateGameHash (HMAC-SHA256 with serverSeed as key). This mismatch means verification will always fail for legitimate results, as the wrong hash is regenerated. Additionally, verifyResultassumes a simple numeric expectedResult and uses hashToNumber, which doesn't align with game-specific derivations (e.g., BigInt for Crash, shuffles for Mines).

Vulnerable Scenario:

1. Play a game (e.g., Dice) and record: serverSeedHash, clientSeed, nonce, result.
2. Reveal serverSeed after rotation.
3. Use verifier path (verifyResult(serverSeed, clientSeed, nonce, expectedResult)).
4. Observe mismatch/failure if verifier uses different hash/derivation.
5. Manually recompute using the actual gameplay function (generateGameHash + dice mapping) and confirm the original result matches, proving verifier mismatch.

Impact

- Users cannot independently verify fairness, resulting in loss of trust and disputes.
- Audit/compliance failure risk: a key fairness control is unreliable/inaccurate.
- Security transparency illusion: the platform appears verifiable but is not, masking regressions or manipulation.

Recommendations

Update verifyResult() to use the exact same hashing function and message format as gameplay:

```
const hash = this.generateGameHash(serverSeed, clientSeed, nonce);
```

Fair Casino:

Fixed

Zealynx:

Verified

[M-05] Game bet amount accepts hexadecimal strings (numeric coercion bypass)

Affected files

*/api/game/**

Description

The backend does not enforce a strict numeric format for wager amounts. Although the UI restricts input to decimal values, the API accepts and processes hex-encoded numeric strings (e.g., "0x10").

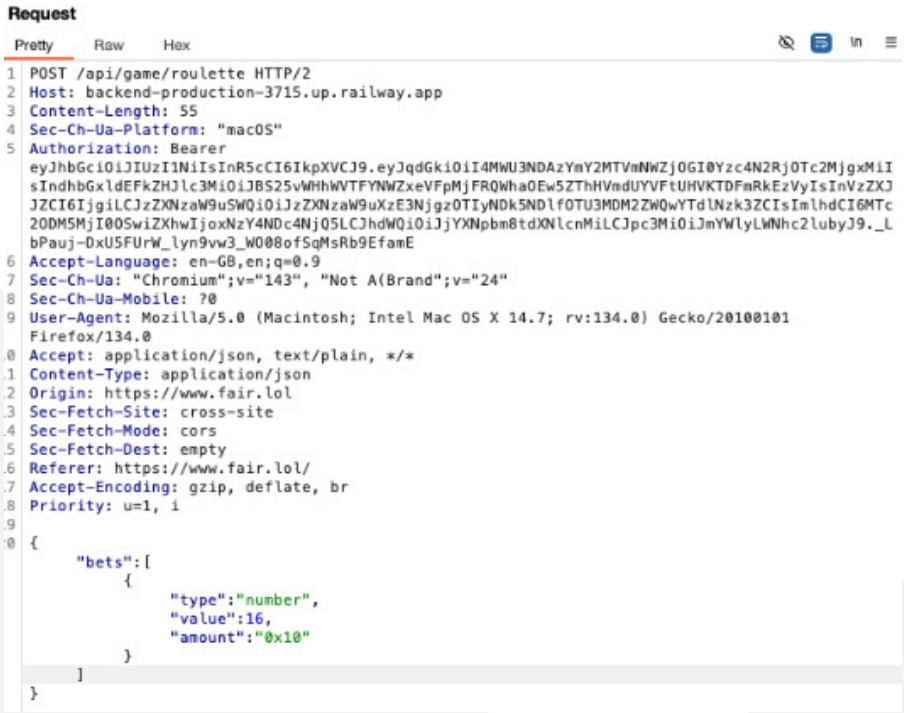
This is risky because:
It demonstrates the backend trusts client input format and relies on JavaScript coercion behavior.

It enables bypassing UI-layer controls and opens the door to other numeric edge cases (scientific notation, Infinity, extremely large values, negative values, etc.), depending on how the server parses/normalizes numbers.

Even if "0x10" is harmless, it's a strong signal that there is no canonicalization policy for money values at the API boundary.

Vulnerable Scenario:

1. Intercept a valid game request (e.g., roulette).
2. Replace a normal decimal amount with a hex string: "amount": "0x10"
3. Forward the request.
4. Observe backend processes it as 16 and returns a normal game result.



Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://www.fair.lol
Access-Control-Expose-Headers: X-Request-Id
Content-Type: application/json; charset=utf-8
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Date: Wed, 14 Jan 2026 12:07:21 GMT
Etag: W/"1c8-Y7AJ7eEtcjAwUQ10GiG1rcsbjUs"
Origin-Agent-Cluster: ?1
Ratelimit-Limit: 100
Ratelimit-Policy: 100;w=5
Ratelimit-Remaining: 98
Ratelimit-Reset: 1
Referrer-Policy: no-referrer
Server: railway-edge
Strict-Transport-Security: max-age=31536000; includeSubDomains
Vary: Origin
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Railway-Edge: railway/europe-west4-drams3a
X-Railway-Request-Id: E2WbzgHXSkmj4V_r0-QtfA
X-Ratelimit-Limit: 30
X-Ratelimit-Remaining: 29
X-Ratelimit-Reset: 1768392443
X-Request-Id: 8f29beee-3ce6-463d-a233-f687aaeac2c3
X-Xss-Protection: 0
Content-Length: 456
```


```
{
    "success":true,
    "gameId":"game_mkdz78dt_3e35bem",
    "result":{
        "number":2,
        "color":"black",
        "wins":[
        ],
        "totalWagered":"16",
        "totalWon":"0"
    },
    "betAmount":"16",
    "winAmount":"0",
    "newBalance":"47.127643",
    "provablyFair":{
        "serverSeedHash":
        "300387f966dd4b2680f138828c1faff9c51ba9c595d53842e0f47a495684e103",
        "clientSeed":"a2cb52c77a205b4f9b8dda790a190e25cd582e88425c449d9dcda91a74beebf5",
        "nonce":72,
        "hash":"0a247089021f027ae8352d063cba574a195ee543d52d580b9e362131ec5a43b1"
    }
}
```


Impact

- Validation bypass: Users can submit values the UI would never allow.
- Potential limit bypass: If min/max checks are implemented against strings or use inconsistent parsing, an attacker may bypass thresholds or precision controls.
- Precision/rounding risk: If decimal precision limits are assumed (e.g., 2 or 6 decimals), alternate encodings can break invariants.

Recommendations

- Enforce strict schema validation on all game endpoints:
- Require amount to be a number type (not a string), OR
- If you intentionally accept strings, enforce canonical decimal with a regex, e.g.:
  ^\d+(\.\d{1,6})?$
- Reject non-canonical formats explicitly:
  "0x..", "1e..", "Infinity", "NaN", "-1", etc.
- Parse money amounts using a strict decimal library (e.g., Decimal.js) with controlled precision, and reject any value that cannot be represented safely.

Fair Casino:

Fixed

Zealynx:

Verified

[M-06] Duplicate start Mines request desynchronizes client/server state, freezing Mines UI until re-auth/refresh

Affected files

*/api/game/**

Description

If the Mines “start game” request is replayed (e.g., resent in Burp/Repeater), the backend rejects the second request with an error indicating an active Mines game already exists. However, after this occurs, the web UI becomes soft-locked:

- The UI shows the Mines game as frozen/unplayable.
- Attempting to start a new Mines game continues to return “already have an active game”.
- The user cannot continue playing or recover state from the UI.
- Recovery requires a full page refresh and re-authentication (or navigation away/back).

This indicates the frontend state machine does not correctly handle the “duplicate start / already active” response. Instead of switching into a “resume active game” state, it gets stuck in an inconsistent state

Vulnerable Scenario:

1. Start a Mines game normally via UI.
2. Intercept the POST /api/game/mines/start request (contains amount, mineCount, gridSize).
3. Send the exact same request again (Burp Repeater).
4. Backend responds with an error like “already have an active mines game”.
5. Return to the UI and observe:
    - Mines game becomes frozen/unplayable
    - starting a new game fails with “already active”
    - cannot recover without refresh + re-auth (or leaving page and coming back)

Likely Root Cause:

Frontend does not reconcile server-side truth after receiving “active game exists”:

- It may clear local game state on error
- It may fail to call “get active mines game” endpoint after a conflict
- It may treat the response as fatal rather than as a recoverable state transition

Request

Pretty Raw Hex

```
POST /api/game/mines/start HTTP/2
Host: backend-production-3715.up.railway.app
Content-Length: 45
Sec-Ch-Ua-Platform: "macOS"
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJlMDExN2I0OTNkYWEzZDg2MjVlYmJhNWI1YTllOTI3ZSI
sIndhbGxldEFkZHJlc3MiOiJBS25vWHhWVTFYNWZxeVFpMjFRQWhaOEw5ZThHVmdUYVFtUHVKTDFmRkEzVyIsInVzZXJ
JZCI6IjgiLCJzZXNzaW9uSWQiOiJzZXNzaW9uXzE3NjgzOTM3OTU3NTNfNjM0MDAzMzc3ZTU5NTJiYiIsImlhdCI6MTc
2ODM5Mzc5NSwiZXhwIjoxNzY4NDgwMTk1LCJhdWQiOiJjYXNpbm8tdXNlcnMiLCJpc3MiOiJmYWlyLWNhc2lubyJ9.v5
5FlgPrW6VqUPGgl-YVuaDMM98pt0jMPq6-7PNd1ZA
Accept-Language: en-GB,en;q=0.9
Sec-Ch-Ua: "Chromium";v="143", "Not A(Brand";v="24"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 14.7; rv:134.0) Gecko/20100101
Firefox/134.0
Accept: application/json, text/plain, */*
Content-Type: application/json
Origin: https://www.fair.lol
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://www.fair.lol/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i

{
    "betAmount":"1",
    "mineCount":1,
    "gridSize":25
}
```

Response

Pretty Raw Hex Render

```
HTTP/2 409 Conflict
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://www.fair.lol
Access-Control-Expose-Headers: X-Request-Id
Content-Type: application/json; charset=utf-8
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Date: Wed, 14 Jan 2026 12:30:44 GMT
Etag: W/"94-L5UFB+RE8qHm2oqfh33gHOZDdMk"
Origin-Agent-Cluster: ?1
Ratelimit-Limit: 100
Ratelimit-Policy: 100;w=5
Ratelimit-Remaining: 99
Ratelimit-Reset: 5
Referrer-Policy: no-referrer
Server: railway-edge
Strict-Transport-Security: max-age=31536000; includeSubDomains
Vary: Origin
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Railway-Edge: railway/europe-west4-drams3a
X-Railway-Request-Id: dIi-7upJT0mQtvvVN8N_Fg
X-Ratelimit-Limit: 30
X-Ratelimit-Remaining: 29
X-Ratelimit-Reset: 1768393846
X-Request-Id: 812bc5d4-b0cc-4344-8d1e-87071310b62c
X-Xss-Protection: 0
Content-Length: 148

{
    "error":"Already have an active Mines game. Cash out or bust first.",
    "code":"MINES_ACTIVE_GAME",
    "requestId":"812bc5d4-b0cc-4344-8d1e-87071310b62c"
}
```

Impact

- Reliable self-DoS / session lock: a user can break their Mines session and must refresh/re-auth to recover.

- Support burden: users will report “Mines is broken / stuck”.

Recommendations

Backend-side hardening (recommended):
Make POST /mines/start idempotent per user:

- If an active mines game exists, return 200 OK with the active game payload (or a consistent “resume” response), not a generic error.
- Or return a dedicated error code (e.g., MINES_GAME_ALREADY_ACTIVE) with the active game ID so the client can recover cleanly.

Frontend-side fix (required):

- On receiving “already active game”:
  - Automatically call GET /api/game/mines/active and load the active game state into UI.
  - Do not clear the local session state;
    - treat as “resume”
- Add a “Resume active game” UI flow if needed.

Fair Casino:

Fixed

Zealynx:

Verified

7.3 Low Severity Findings

[L-01] Fairness “Verifier” can generate “Verification Successful” results for games that never occurred (not bound to game history)

Affected files

backend/src/domain/provably-fair.ts

frontend/src/components/provably-fair/FairnessCenter.tsx

The UI:

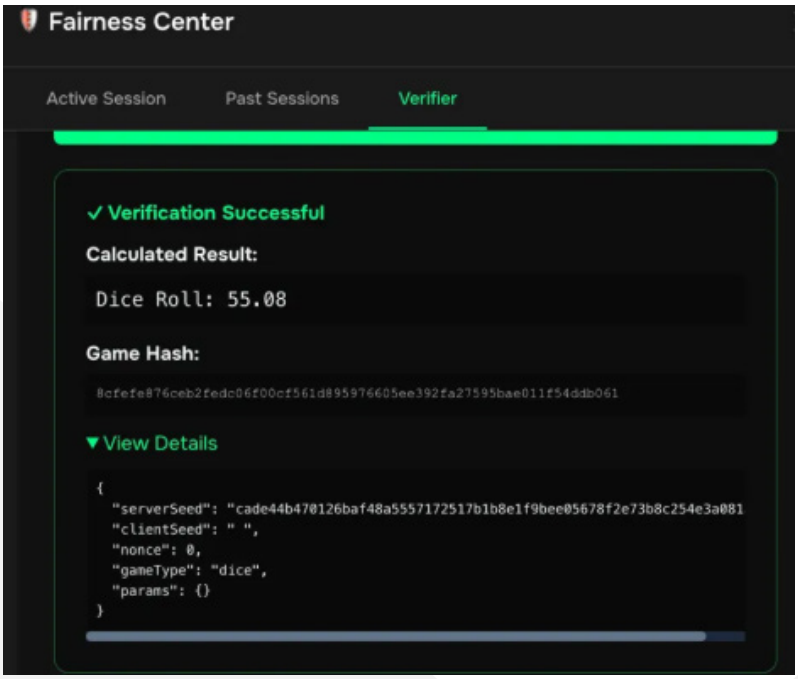
- allows manual entry of seeds and nonce
- does not require selecting a game from history
- displays “ Verification Successful” regardless of gameplay state

Description

The Fairness Center verifier computes and returns a “ Verification Successful” result for any provided serverSeed, clientSeed, and nonce, even when Games Played = 0 and no wager/game execution occurred. However, the verifier is not bound to authoritative game records and allows successful verification of results even when no game was actually played. This behavior can mislead users into believing a real game outcome has been verified, when in fact the verifier is simply recomputing the deterministic output for arbitrary inputs.

With Games Played = 0, the verifier returns success and a valid dice roll for:

- serverSeed = cccbde...
- clientSeed = " " (whitespace)
- nonce = 0
- Result: Dice Roll 90.34, hash e741b0...



Recommendations

Make verifier operate in two modes:

History-bound verification (default): user selects a real game from history; verifier recomputes and compares against recorded outcome.

Manual simulation (optional): clearly labeled “Simulate result” with disclaimers (“does not prove a bet occurred”).

Fair Casino:

Fixed

Zealynx:

Verified

[L-02] Insecure Randomness in Session ID Generation

Affected files

backend/src/domain/provably-fair.ts

Description

Session IDs are generated using Math.random(), which is not cryptographically secure and can be predictable (especially in Node.js, where it's pseudo-random with low entropy). Combined with Date.now() (millisecond-precision timestamp), an attacker with timing knowledge could guess/collide IDs. While sessionIDs are internal (used for repo storage and frontend sync), exposure (e.g., via logs or responses) could enable session hijacking or enumeration.

Recommendations

Replace with crypto-secure random:

```
import crypto from 'crypto';
const randomPart = crypto.randomBytes(8).toString('hex');
const sessionId = `${userId}-${Date.now()}-${randomPart}`;
```

Fair Casino:

Fixed

Zealynx:

Verified

[L-03] Reflected Request Path in Error Response (Unnormalized Route Handling)

Affected files

GET /api/ (check all API routes)*

Description

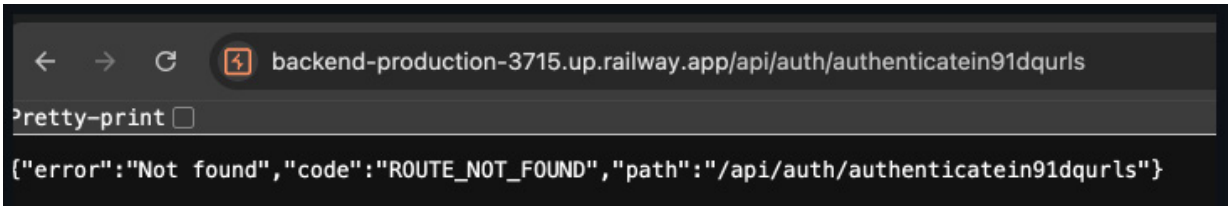
When an invalid or malformed authentication route is requested, the backend returns a 404 ROUTE_NOT_FOUND response that includes the user-supplied path verbatim in the JSON body.

This indicates that:

Route matching does not normalize or strictly validate expected paths before error handling.

Arbitrary suffixes appended to sensitive routes (e.g. /api/auth/authenticate*) are accepted and reflected back to the client.

While no script execution or header injection was observed, reflecting user-controlled path data in structured responses is generally discouraged, especially on authentication endpoints.



Recommendations

Do not reflect raw request paths in error responses. Replace with a static value, e.g.:

```
{
  "error": "Not found",
  "code": "ROUTE_NOT_FOUND"
}
```

Fair Casino:

Fixed

Zealynx:

Verified