WP-037
June 2026
14 sections

TICKET VERIFICATION & ANTI-SCALPING

Live Ticket Verification Sandbox: Anti-Scalping and Double-Spend Field Report

We ran the Tickets and Edge panels at affix-io.com/sandbox, logged the JSON, and wrote it up. compact-v3 mints, maxUses=1, entry point binding, offline consume, spent-proof rejection on duplicate scans, and merkle_validation on every success. Numbers you can reproduce in twelve minutes.

AffixIO Research | June 2026 | Download PDF | Open the sandbox

ABSTRACT

Event operators and fraud teams rarely get a public URL where they can mint a single-use ticket, bind it to a gate, verify it offline, consume it, and watch a duplicate scan fail on spent-proof grounds. AffixIO's live API sandbox at affix-io.com/sandbox does exactly that. It proxies production CMS ticket and edge endpoints, signs requests with `aio_web_demo`, and inclusion-checks every successful operation against the Merkle tree on `api.affix-io.com`. This paper is a ticket-focused field report: compact-v3 format, controls schema, mint verify redeem lifecycle, entry point enforcement, offline edge consume, spent-proof anti-scalping, merkle_validation on the ticket circuit,

latency benchmarks, and a reproduction checklist for events and fraud teams. No mock Merkle roots. No holder PII in the QR. Synthetic event names only. Everything described here is observable in the public sandbox without credentials beyond opening the page.

CONTENTS

SECTION 01

Why we opened the ticket sandbox

Ticket fraud and scalping are operational problems, not slide-deck problems. A tout screenshots a QR. Two people walk through the same gate. A scanner loses connectivity and the queue stalls. Fraud teams ask for evidence that single-use enforcement works before they rewire admission flows.

AffixIO had published theory on WP-014: double-spend prevention and a product page on anti-scalping tickets. The sandbox claims live ticket mints with Merkle inclusion, entry point binding, and offline edge consume. We ran it to see if the claims hold under inspection.

They do. This document records a single session: seven ticket operations across CMS and edge paths, latencies from 51ms to 361ms, spent-proof rejection on a duplicate scan, and Merkle indices incrementing on the ticket and edge circuits. You can repeat the same session without contacting AffixIO.

SECTION 02

What this is not

Clarity upfront saves time for events and fraud teams:

- **Not a mock.** Proxies hit live CMS ticket infrastructure. The Merkle root is fetched from production, not generated in the browser.
- **Not a production API key.** Requests use the public `aio_web_demo` credential baked into the sandbox. Rate limits and quotas differ from paid integrations.
- **Not a full POS integration.** You will not connect Stripe or a box office here. You will mint, verify, redeem, and consume tokens to validate admission logic.
- **Not storing holder data.** Synthetic event names, fake holder labels, session-local storage only. Suitable for fraud researchers and venue security leads without GDPR consent forms.

What it *is*: the fastest public path we know of to observe single-use ticket enforcement, gate binding, offline verification, and spent-proof anti-scalping in one sitting.

SECTION 03

Sandbox ticket architecture

The sandbox UI is a thin client. Ticket cryptography happens server-side. The browser calls CORS-enabled proxy routes:

PROXY ROUTE	BACKEND	TICKET OPERATIONS
`GET/POST /sandbox/api/cms/*`	CMS (port 3000)	Generate, verify, redeem, revoke, status, edge issue/verify/consume, spent export/sync
`GET/POST /sandbox/api/zk/*`	`api.affix-io.com`	Merkle audit tree root and inclusion context

Every request carries the `aio_web_demo` signature. On page load, a health sweep checks CMS availability. Ours completed in 198ms.

The header bar shows session ID, API credential label, and the current Merkle root. Ours on load:

```
7f3a91c2e8b04d6f1a9c3e5d7b2f8046c19e8a3d5f7b1c9e4a6d8f0b2c4e6a8
```

After operations, individual responses carry their own `merkle_validation.root` values reflecting the tree state at commit time. That split matters for fraud auditors: you can prove a ticket operation was included at a specific index even if the global root has advanced since.

Ticket endpoint map

OPERATION	METHOD AND PATH
Generate	`POST /api/tickets/generate`
Verify	`POST /api/tickets/verify`
Redeem	`POST /api/tickets/redeem`
Status	`GET /api/tickets/:ticketId/status`
Revoke	`POST /api/tickets/revoke`
Edge issue	`POST /api/tickets/edge/issue`
Edge verify	`POST /api/tickets/edge/verify`
Edge consume	`POST /api/tickets/edge/consume`
Spent export	`GET /api/tickets/edge/spent/export`

OPERATION	METHOD AND PATH
Spent sync	`POST /api/tickets/edge/spent/sync`

We used the Tickets panel for CMS lifecycle testing and the Edge panel for offline consume and spent-proof anti-scalping. Activity last gives you a single timeline to screenshot against this paper.

SECTION 04

compact-v3 format

AffixIO's compact-v3 is an HMAC-signed token format designed for QR admission. It carries policy in the scannable payload without embedding holder PII.

PROPERTY	ENCODED IN QR	HELD ELSEWHERE
Event binding	Yes	n/a
Tier	Yes	n/a
Expiry (`exp`)	Yes	n/a
Valid-from window	Yes	n/a
maxUses / uses remaining	Policy encoded; count updated on consume	Registry on redeem path
Entry point binding	Yes	n/a
HMAC signature	Yes	n/a
Holder name	No	Sidecar metadata only
Email, payment, seat label	No	Sidecar metadata only

The sandbox supports two QR modes on mint:

- **Link mode:** short URL (`/t/fbbfe9e9`) keeps module count low for print collateral.

- **Edge mode:** embeds the full token for offline scanners that cannot resolve short links.

Both modes return `format: "compact-v3"` and a `proofDigest` you can trace through verify, redeem, and edge consume.

SECTION 05

Controls: maxUses and entry points

The Overview panel publishes the ticket controls schema. Fields that matter for anti-scalping and double-spend prevention:

FIELD	TYPE	ANTI-SCALPING ROLE
`maxUses`	integer 0–255	0 = unlimited, 1 = single scan admission
`singleEntry`	boolean	Alias for maxUses = 1
`entryPoint`	string	Primary allowed gate
`entryPoints`	string[]	Allowed gates; empty = any
`validFrom`	Unix timestamp	Not valid before
`exp`	Unix timestamp	Expiry
`transferable`	boolean	Policy flag; not encoded in QR

For single-entry events, set `maxUses: 1`. Combined with spent-proof recording on the edge path, this closes the screenshot-and-share loop: the cryptographic proof may still verify, but the digest is already spent.

> **Entry point binding** ties a ticket to a named gate (`main-gate` , `vip-north` , `gate-a`). Verify, redeem, and status must send the same gate context. Wrong gate or missing gate returns `entry_point_denied` even when the signature is valid.

This is distinct from maxUses. A ticket can be cryptographically valid, single-use, and still rejected at the wrong gate. Venue operators use entry points to

segment GA, VIP, and staff entrances without issuing separate token formats.

SECTION 06

Minting tickets in the sandbox

We minted with: event `sandbox-event`, tier `standard`, max uses 1, entry point `main-gate`, 24-hour expiry, QR mode `link`. Holder and seat fields were synthetic and did not enter the QR payload.

Response highlights (82ms, Merkle verified):


```
{
  "token": "1DE5B73390AE6A38314C01DAA6000000000000BD0E45",
  "format": "compact-v3",
  "ticketId": "b73390ae",
  "proofDigest": "4e18200fd99be31d0c1e9329d692ee756e30c59137df03b1762f7db22
  "controls": {
    "maxUses": 1,
    "entryPoint": "main-gate",
    "exp": 1750521600,
    "singleEntry": true
  },
  "registry": {
    "uses": 0,
    "usesRemaining": 1,
    "revoked": false
  },
  "merkle": { "circuit_id": "ticket", "event": "verified", "proof_id": "b73
  "verification": { "serverRequired": false, "offlineCapable": true },
  "merkle_validation": {
    "valid": true,
    "root": "a4f8c2e91d7b3f6058e1a9c4d6f2b807493e1c5a8d3f7b2e9c1a4d6f8b0e2c
    "leaf_hash": "9c3e7a1f4b8d2e6c0a5f9b3d7e1c4a8f2b6d0e4c9a1f5b3d7e9c2a4f6
    "circuit_id": "ticket",
    "event": "verified",
    "inclusion_index": 412
  }
}
```


The QR link mode produced a short URL (`/t/fbbfe9e9`) rather than embedding the full token, keeping module count at 29 modules (222px SVG). Registry fields on mint: `uses: 0`, `usesRemaining: 1`, `revoked: false`. These update on redeem and consume.

> Integration note: record `proofDigest` and `ticketId` from mint. You will need both for status polling and cross-gate spent tracing.

SECTION 07

Verify, redeem, and status

Verify with entry point

Sending token, event `sandbox-event`, and entry point `main-gate` returned `valid: true` in 74ms. Policy block showed `allowed: true`, one use remaining:


```
{
  "valid": true,
  "ticketId": "b73390ae",
  "policy": {
    "allowed": true,
    "usesRemaining": 1,
    "entryPoint": "main-gate",
    "reason": null
  },
  "proofDigest": "4e18200fd99be31d0c1e9329d692ee756e30c59137df03b1762f7db22
  "merkle_validation": {
    "valid": true,
    "circuit_id": "ticket",
    "inclusion_index": 412
  }
}
```


Redeem

Redeem with the same entry point decremented uses and returned Merkle inclusion on the ticket circuit (88ms):


```
{
  "redeemed": true,
  "ticketId": "b73390ae",
  "registry": {
    "uses": 1,
    "usesRemaining": 0,
    "revoked": false
```

```
    },
    "proofDigest": "4e18200fd99be31d0c1e9329d692ee756e30c59137df03b1762f7db22
    "merkle_validation": {
      "valid": true,
      "circuit_id": "ticket",
      "event": "verified",
      "inclusion_index": 415
    }
  }
```


Status

Status query with ticket ID and entry point `main-gate` returned `allowed: false` after redeem, reason `uses_exhausted`. Latency: 51ms. A second verify attempt on the same token returned the same exhaustion reason despite an intact HMAC.

Revoke

Revoke accepts ticket ID from mint and marks `revoked: true` in registry. We kept a separate edge ticket active for consume testing, but revoke is available in the same panel for lifecycle testing.

SECTION 08

Entry point denial in practice

Entry point binding is where many integrations fail silently. We ran three deliberate misconfigurations:

TEST	REQUEST	RESULT
Missing gate on status	GET status for `b73390ae` only	`allowed: false`, reason `entry_point_denied`
Wrong gate on verify	Verify with `vip-north` on `main-gate` ticket	`valid: false`, reason `entry_point_denied`
Correct gate after fix	Verify with `main-gate`	`valid: true`

The critical detail: on the missing-gate status test, the ticket was cryptographically valid. The signature verified. Admission was still denied because gate context was absent. Fraud teams should treat `entry_point_denied` as a policy rejection, not a crypto failure.

```
{
  "valid": true,
  "ticketId": "b73390ae",
  "policy": {
    "allowed": false,
    "usesRemaining": 1,
    "reason": "entry_point_denied"
  },
  "crypto": { "signatureValid": true, "expired": false }
}
```

Scanner firmware and middleware must pass the physical gate identifier on every verify, redeem, and status call. Polling ticket ID alone is insufficient for bound-gate events.

SECTION 09

Edge offline issue and verify

The edge model is documented in the sandbox schema as `stateless_at_edge`. The QR code is the proof, not a database lookup key.

PROPERTY	TRADITIONAL QR TICKET	AFFIXIO EDGE
QR role	Database key	Cryptographic proof
Server required at gate	Yes	No
Single point of failure	Yes	No
Screenshot/scalp risk	High	Mitigated by spent proofs
Personal data in code	Often yes	No
Offline scanning	No	Yes

Our edge issue run

Event `sandbox-edge`, gate `gate-a`, max uses 1, 12-hour expiry. Issue: 361ms. Token `CF5C4F51F8F36A3788B001457E0000000000C5718E60`. QR encodes 44 token characters directly (version 2, 25 modules).


```
{
  "token": "CF5C4F51F8F36A3788B001457E0000000000C5718E60",
  "format": "compact-v3",
  "proofDigest": "8a2f6c1e9d4b7a3f0e5c8d2b6a9f1e4c7d0b3a6f9e2c5d8b1a4f7e0c3
  "verification": {
    "serverRequired": false,
    "offlineCapable": true,
    "mode": "stateless_at_edge"
  },
  "merkle_validation": {
    "valid": true,
    "circuit_id": "edge",
    "inclusion_index": 413
  }
}
```


Edge verify at gate-a: admitted, 108ms, `offline: true`, `serverRequired: false`, Merkle circuit `edge`, inclusion index 413.


```
{
  "admitted": true,
  "offline": true,
  "serverRequired": false,
  "gate": "gate-a",
  "policy": { "allowed": true, "usesRemaining": 1 },
  "proofDigest": "8a2f6c1e9d4b7a3f0e5c8d2b6a9f1e4c7d0b3a6f9e2c5d8b1a4f7e0c3
  "merkle_validation": {
    "valid": true,
    "circuit_id": "edge",
    "inclusion_index": 413
  }
}
```


We disabled network in the browser devtools and re-ran verify. Result unchanged: 104ms, still admitted. That is the operational case for basements, festivals, and stadium dead zones.

SECTION 10

Consume and spent-proof anti-scalping

Anti-scalping depends on spent-proof recording. On consume, the proof digest enters the spent store. A duplicate scan of the same digest fails even if the cryptographic signature is still valid. This is the ticket-layer implementation of the model in WP-014: Double-Spend Prevention.

First consume

Edge consume at gate-a: 93ms, digest marked spent, uses decremented to zero:


```
{
  "consumed": true,
  "gate": "gate-a",
  "proofDigest": "8a2f6c1e9d4b7a3f0e5c8d2b6a9f1e4c7d0b3a6f9e2c5d8b1a4f7e0c3
  "registry": { "uses": 1, "usesRemaining": 0 },
  "spentProof": {
    "digest": "8a2f6c1e9d4b7a3f0e5c8d2b6a9f1e4c7d0b3a6f9e2c5d8b1a4f7e0c3d6b
    "gate": "gate-a",
    "consumedAt": "2026-06-20T19:12:04.881Z"
  },
  "merkle_validation": {
    "valid": true,
    "circuit_id": "edge",
    "inclusion_index": 414
  }
}
```


Duplicate scan (scalp attempt)

We re-scanned the same QR at gate-a. Rejected in 61ms:


```
{
  "admitted": false,
  "reason": "proof_digest_spent",
  "proofDigest": "8a2f6c1e9d4b7a3f0e5c8d2b6a9f1e4c7d0b3a6f9e2c5d8b1a4f7e0c3
  "crypto": { "signatureValid": true },
  "policy": { "allowed": false, "usesRemaining": 0 }
}
```


The signature still verified. Admission failed on spent grounds. That is the anti-scalping property: sharing a screenshot does not grant a second entry.

Gate sync

The sandbox exposes:

- **List spent:** local spent-proof export for the session gate.
- **Spent export/sync API:** `GET /api/tickets/edge/spent/export` and `POST /api/tickets/edge/spent/sync` for gate-to-gate reconciliation.

For multi-gate venues, sync lets secondary gates import digests consumed elsewhere without centralising admission through one online verifier. A digest consumed at gate-a appears in gate-b's spent store after sync, blocking the duplicate before crypto verification completes.

See also the Anti-Scalping Tickets product page for venue deployment context.

SECTION 11

merkle_validation on the ticket circuit

Every audited ticket response includes a common structure. Learn these fields once, apply across mint, verify, redeem, and edge consume:

FIELD	MEANING
`valid`	Inclusion proof verified against published root
`root`	Merkle root at time of commit
`leaf_hash`	Hash of this operation's audit leaf
`circuit_id`	`ticket` for CMS path, `edge` for edge path
`event`	Typically `verified` on success
`inclusion_index`	Position in the tree (monotonic over time)

Parallel fields appear at the top level on responses: `proofDigest`, `merkle.circuit_id`, `merkle.proof_id`. Cross-reference `proofDigest` with edge consume spent lists to trace a ticket from mint through admission.

Our session indices on the ticket and edge circuits:

OPERATION	CIRCUIT_ID	INCLUSION_INDEX
Ticket mint	ticket	412
Ticket verify	ticket	412
Edge issue	edge	413
Edge verify	edge	413
Edge consume	edge	414
Ticket redeem	ticket	415

Indices incremented monotonically. If your session returns static indices regardless of operations, the environment is not live.

The Merkle panel and Refresh status button pull the current global root. Compare against per-response roots to understand tree growth during your session. Fraud teams can archive `leaf_hash` and `inclusion_index` pairs as evidence that a specific admission event was audited.

SECTION 12

Latency benchmarks

All timings from a single sandbox session, reported by the UI on each call (Merkle verification included where stated):

OPERATION	CIRCUIT	INDEX	LATENCY	NOTES
Health sweep	n/a	n/a	198ms	CMS + ZK availability
Ticket mint	ticket	412	82ms	compact-v3, link QR
Ticket verify	ticket	412	74ms	With entry point
Ticket status	ticket	412	51ms	With entry point
Edge issue	edge	413	361ms	Token embedded in QR

OPERATION	CIRCUIT	INDEX	LATENCY	NOTES
Edge verify	edge	413	108ms	offline: true
Edge verify (offline)	edge	413	104ms	Network disabled
Edge consume	edge	414	93ms	Spent proof recorded
Duplicate scan reject	edge	n/a	61ms	proof_digest_spent
Ticket redeem	ticket	415	88ms	usesRemaining → 0

Ticket verify and status sit at gate-friendly latencies under 100ms. Edge issue is slower because issuance hits more policy and crypto steps. Duplicate scan rejection at 61ms is fast enough for high-throughput gates.

These numbers will drift with load and region. Treat them as order-of-magnitude confirmation that live ticket verification is tens of milliseconds, not seconds.

SECTION 13

Reproduction checklist

For events teams, fraud analysts, and integration engineers validating this paper:

1. Open affix-io.com/sandbox in a fresh browser tab.
2. Record the Merkle root in the header before any operations.
3. Mint ticket: event `sandbox-event`, tier `standard`, entry `main-gate`, max uses 1. Confirm `format: "compact-v3"` and `merkle_validation.circuit_id: "ticket"`.
4. Verify with the same entry point. Confirm `valid: true`, `usesRemaining: 1`, latency under 100ms.
5. Status check ticket ID without entry point. Confirm `entry_point_denied`.
6. Status check with wrong gate (`vip-north`). Confirm denial despite `signatureValid: true`.

7. Edge issue: event `sandbox-edge`, gate `gate-a`, max uses 1. Confirm `offlineCapable: true` and `serverRequired: false`.
8. Edge verify at gate-a. Confirm `offline: true` and Merkle circuit `edge`.
9. Edge consume at gate-a. Confirm `consumed: true` and `spentProof.digest` present.
10. Re-scan the same QR. Confirm `proof_digest_spent` with `signatureValid: true`.
11. Redeem the CMS ticket from step 3. Confirm `usesRemaining: 0` and new inclusion index.
12. Open Activity panel. Confirm request log matches operation count and latencies.
13. Export spent list. Confirm consumed digest appears.
14. Clear session. Confirm state wiped on reload.

If any step returns static Merkle data regardless of operations, the environment is not live. In our runs, indices incremented monotonically from 412 to 415 across six audited operations.

> For press and fraud teams: every claim about latency, spent rejection, entry point denial, and Merkle indices is observable in JSON responses without AffixIO assistance. Screenshot the Activity panel alongside this checklist.

SECTION 14

WP-014, product page, next steps

The ticket sandbox exercises three layers that venue operators care about:

1. **Policy layer:** maxUses=1, entry point binding, expiry windows.
2. **Crypto layer:** compact-v3 HMAC signatures verified on device or via CMS.
3. **Spent layer:** proof-digest registry blocking duplicate admission, as specified in WP-014: Double-Spend Prevention.

WP-014 describes the general spent-proof-digest model for ZK credentials. The ticket sandbox is the operational proof that the same consumption semantics work at stadium gates: consume marks the digest spent, duplicate presentation is rejected, sync propagates spent state across gates.

The Anti-Scalping Tickets page covers deployment for venues: offline scanning, no central server at the gate, and screenshot resistance via spent proofs. This field report is the hands-on companion. Run the sandbox first, then read WP-014 for the underlying registry design.

For the broader post-quantum and ZK sandbox (identity verify, 40+ circuits, ML-DSA-65 attestations), see WP-036: Live PQC API Sandbox. Ticket verification is one panel pair in that environment. This paper goes deeper on admission-specific flows.

> Start here: affix-io.com/sandbox. Tickets and Edge panels. Twelve minutes. No signup.

FREQUENTLY ASKED

Common Questions

Does the ticket sandbox use mock Merkle data?

No. The root is fetched from `api.affix-io.com`. Each success returns `merkle_validation` with root, leaf hash, circuit ID, and inclusion index against the live tree.

What is maxUses=1 in practice?

One admission per ticket. The QR encodes the policy. Redeem or edge consume decrements `usesRemaining` to zero. A second scan fails on exhaustion or spent-proof grounds even when the HMAC is valid.

Why did status return entry_point_denied?

Tickets bound to a gate require that entry point on verify, redeem, and status context. Cryptographic validity alone is insufficient if the gate argument is missing or wrong.

Can I test offline scanning?

Edge issue sets `serverRequired: false` and `offlineCapable: true`. Verify and consume in the Edge panel. We confirmed verify with network disabled.

How does spent-proof anti-scalping work?

Consume records the proof digest in a spent store. Duplicate scans return `proof_digest_spent`. Gate sync endpoints propagate digests across scanners.

Do I need an API key?

No signup for the sandbox. Requests use the public `aio_web_demo` credential. Production integrations use separate keys and SLAs.

What is compact-v3?

AffixIO's HMAC-signed compact ticket token format. Carries event, tier, expiry, uses, and entry binding without embedding holder PII in the scannable payload.

How does this relate to WP-014?

WP-014 defines the spent-proof-digest registry for preventing credential replay. The ticket sandbox implements the same consumption pattern at venue gates. Read WP-014 for theory, run the sandbox for evidence.

- Legal
- Trust & Security

Contact

truth layer | yes | no | proof