A Layered Model of Modern Web Bot Protection and the Structural Limits of Its Circumvention

Sudipto Chandra
Independent Researcher

June 2026

Abstract

Commercial bot-mitigation services no longer base their decisions on a single signal. Since roughly 2022 they have converged on what we term composite trust scoring, in which a request is admitted only when it satisfies many independent detectors at once. This paper analyses that architecture layer by layer, using Cloudflare as the running example because it is the provider a scraper is most likely to encounter. For each detection mechanism we describe how it works, the techniques available to circumvent it, and, where applicable, why no reliable circumvention exists. Two ideas organise the analysis. The first is a simple observation about composite scoring: because admission behaves as a near-conjunction of independent checks, the probability that a given strategy evades detection is bounded, to a good approximation, by its weakest layer. One consequence is that improving a layer the strategy already fails buys nothing once a different layer has become the binding constraint. The second idea concerns what each detector reads. Some detectors inspect an artifact the client emits at connection time, such as a TLS `ClientHello`, a sequence of HTTP/2 frames, or a header order; these are reproducible in principle, because the artifact carries no secret. Others inspect a property the client must hold continuously and cannot fabricate on demand, such as accumulated per-zone behavioural history or a private signing key; these resist forgery by construction. We argue that this emit/possess distinction predicts circumvention difficulty more reliably than the usual network-versus-browser split. The paper maps the detection stack onto the framework as a sequence of nineteen layers, surveys the open-source tooling that is current as of mid-2026 (`curl_cffi`, `nodriver`, Camoufox, Patchright), and examines the layers that read a possessed property: per-zone behavioural models, which can only be approached by genuinely accumulating the history they inspect, and cryptographic agent identity (Web Bot Auth, built on RFC 9421), which cannot be forged at all where a site requires it. A closing section discusses the legal and ethical limits on legitimate practice.

Keywords: web scraping, bot detection, TLS fingerprinting, browser automation, HTTP message signatures, Cloudflare.

1 Introduction

Automated retrieval of web content supports a range of legitimate activities, including price monitoring, academic research, accessibility tooling, archival, and competitive intelligence. The environment in which it operates has changed considerably. A decade ago, avoiding detection meant little more than rotating a few IP addresses and setting a plausible `User-Agent`. A modern mitigation provider may instead evaluate a dozen or more largely independent signals on every request and fold them into a single trust score.

Cloudflare is the provider a scraper is most likely to meet. Public estimates of its share of the bot-mitigation market vary, but it is consistently reported as the largest single provider, and Cloudflare states that automated traffic on its network already runs to billions of requests per week, with its own leadership expecting bot traffic to overtake human traffic later this decade [1, 2]. For anyone operating a data pipeline, knowing how this protection is assembled, and where it cannot be circumvented at all, is of practical value.

The paper makes three contributions: a composite-scoring model that frames evasion as a non-compensatory conjunction of detectors, from which an asymmetry between attacker and defender effort follows (Section 2); the emit/possess distinction, which we argue predicts circumvention difficulty better than the usual network-versus-browser split (Section 2.2); and a consolidated, layer-by-layer survey of current detection mechanisms and the tooling that addresses them (Sections 3 and 4). The remainder of the paper applies this material: Section 5 turns it into a small set of reference patterns for practitioners, Section 6 records techniques that no longer work, and Section 7 sets out the legal and ethical limits on legitimate practice.

The scope is deliberately narrow. We consider only the retrieval of publicly accessible content, and exclude authentication bypass, credential abuse, and any activity that circumvents access controls protecting non-public data.

2 The Composite-Scoring Model

A modern mitigation engine runs a set of detectors $\{d_1, \ldots, d_n\}$ on each request and combines their outputs into a single trust score. The score determines one of three outcomes: a high score returns the content with a `200 OK`; an intermediate score returns an inter-

active challenge, either a JavaScript interstitial or a CAPTCHA; a low score returns a block, typically a `403` or a Cloudflare `1020` firewall response. The pipeline is thus request, parallel detectors, composite score, then a banded decision (Figure 1).

2.1 A Bound on Evasion Probability

The bound that follows depends on one assumption about how the score behaves. We treat the score as approximately non-compensatory: a strong result on most detectors does not fully offset a single strongly anomalous one. Real scoring is not perfectly non-compensatory, but in practice a sufficiently anomalous signal, such as a datacenter IP or a non-browser TLS fingerprint, tends to dominate the outcome, which is what the assumption captures. Under it, let p_i be the probability that some adversarial strategy passes detector d_i. Admission then requires passing every detector that is individually capable of triggering a block, so the probability of evading detection overall is bounded by the weakest such layer:

$$P_{\text{evade}} \lesssim \min\big(p_1, p_2, \ldots, p_n\big). \tag{1}$$

Equation (1) is elementary: a conjunction is no more likely than its least likely conjunct, so the joint probability of passing every detector cannot exceed the smallest individual pass probability. Its value here is interpretive rather than mathematical. It explains why no single technique suffices, and it has two practical consequences. If a strategy already fails some detector, repairing a different detail it also fails does not raise P_{evade} until the original failing detector is no longer the minimum; effort spent elsewhere is wasted. Conversely, a defender who adds even one detector with $p_i \approx 0$ drives P_{evade} toward zero whatever the strategy scores on the rest. The two sides of this bound are not symmetric in effort: an attacker must raise the minimum across all active layers, whereas a defender need only introduce one layer that is close to impassable.

2.2 What Each Detector Reads: Emit versus Possess

Detection is often divided into network-level and browser-level checks. That division describes where a check sits, but it does not predict how hard the check is to defeat. A more useful question is what kind of thing each detector reads.

A detector may read an *emitted artifact*: something the client transmits at connection time, such as the bytes of a TLS `ClientHello`, the ordering of HTTP/2 frames, or the sequence of request headers. The artifact carries no secret and records no history, so anything a genuine browser emits can, in principle, be reproduced by a client built to do so. Detectors of this kind are reliable against naive clients but offer little against a faithful imitator.

Alternatively, a detector may read a *possessed property*: something the client must hold continuously and cannot fabricate on demand, such as accumulated per-zone behavioural history or a private cryptographic key. These resist forgery not through obscurity but because reproducing them amounts to actually being what the check tests for.

This distinction has a direct architectural consequence. The transport-and-header layers (Layers 2 to 5: TLS fingerprinting, the post-quantum key share, HTTP frame fingerprinting, and header order) all read emitted artifacts, and they are correlated: a client built to reproduce one browser's network stack tends to pass all of them at once. They therefore do not act as four independent barriers. In the bound of Eq. (1) they behave as a single term, so adding another emit-reading check of the same kind barely moves P_{evade}, whereas adding one possess-reading check can lower it sharply. The rest of the paper uses the emit/possess label to indicate, for each layer, whether durable resistance is possible at all. Figure 2 places the layers along this axis.

3 The Detection Layers

We define nineteen detection layers and treat each separately, surveying them in the order they appear over a request's lifetime, from network transport up to cryptographic identity. For each layer we describe the mechanism precisely, classify it on the emit/possess axis, and give the circumvention options that remain viable in 2026. Where consecutive layers share a single circumvention, as the four transport-and-header layers share `curl_cffi`, we say so explicitly rather than merging them, because their detection mechanisms and emit/possess characters differ even when the practical countermeasure does not.

The numbering is an organising device of our own, adopted for exposition. The underlying mechanisms, product names such as Turnstile and Bot Fight Mode, and fingerprinting techniques such as JA3 and JA4 are drawn from Cloudflare's documentation and the wider security literature; they are not presented here as an official taxonomy.

3.1 Layer 1: IP Reputation

Every source address is scored against a global threat-intelligence database that draws on datacenter ASN ranges, Tor exit lists, the reputation of residential-proxy pools, and carrier-grade NAT (CGNAT) classification for mobile carriers. In the emit/possess framing this layer is mixed. The address itself is freely chosen and so is emitted, but the reputation attached to it is a possessed property built up over time, which an adversary can rent but not fabricate. Its remaining difficulty is economic rather than technical. Datacenter ranges are cheap to block because little legitimate human traffic originates there. Mobile-carrier ranges are close to unblockable, because a single CGNAT address fronts thousands of real subscribers and blocking it would cause widespread collateral damage. The practical ranking, from most to least useful, is mobile carrier, then real

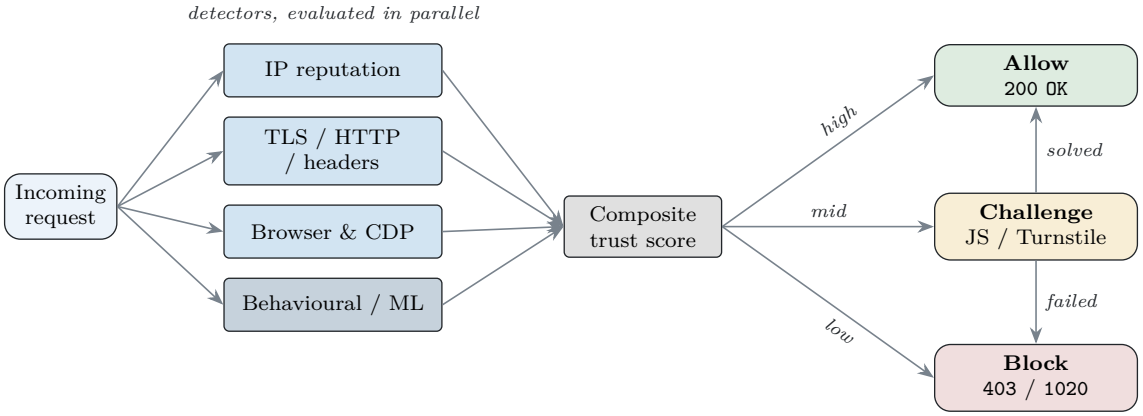

Figure 1: The composite-scoring pipeline. Detectors are evaluated in parallel and combined into a single trust score, which is mapped to one of three outcomes. An intermediate score yields a challenge, whose result then routes to allow or block. The four detector boxes group the layers of Section 3 by kind, for legibility. Shading follows the emit/possess palette used throughout: lighter cells read emitted artifacts, the darker cell reads a possessed property.

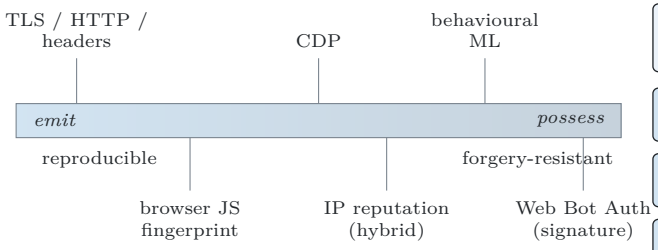

Figure 2: The emit/possess spectrum. Checks that read an emitted artifact sit toward the left and are reproducible by a faithful imitator; checks that read a possessed property sit toward the right and resist forgery. IP reputation and CDP detection fall in between. The transport-and-header checks cluster at the left because a single impersonation client reproduces them together.

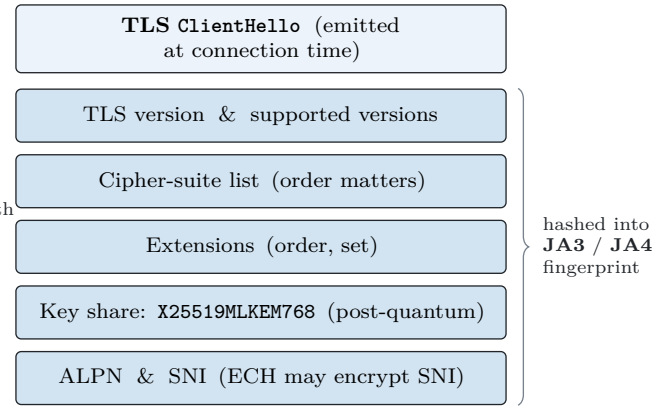

Figure 3: Anatomy of the `ClientHello` that a TLS fingerprinter reads. The version list, ordered cipher suites, extension set, post-quantum key share, and ALPN are combined into a JA3 or JA4 hash. None of these fields carries a secret, so a client engineered to reproduce a target browser's exact bytes produces a matching fingerprint, which is why all rows read as emitted artifacts. The post-quantum key share is called out separately as Layer 3 because its absence dates a profile: a client claiming to be current Chrome while omitting it is anomalous on that basis alone.

residential, then static ISP, with datacenter and Tor addresses reliably blocked.

3.2 Layer 2: TLS Handshake Fingerprinting (JA3/JA4)

Every HTTPS client opens with a TLS `ClientHello` that exposes, in cleartext, the offered TLS version, the ordered cipher-suite list, the extension set and order, and the supported elliptic curves. JA3 [3] hashes these fields into a single value; the more recent JA4 [4] produces a multi-part fingerprint that additionally covers ALPN and SNI presence and separates the cipher and extension hashes, which makes it harder to spoof. This layer reads a purely *emitted* artifact: the handshake carries no secret, so it records exactly what the client chose to send. Ordinary Python clients such as `requests`, `httpx`, and `aiohttp` emit a handshake that resembles no browser and are identified within the first round trip. Figure 3 shows the fields a fingerprinter reads. Because the artifact is reproducible, a client engineered to send a target browser's exact `ClientHello` bytes produces a matching fingerprint; the circumvention is shared with the rest of this group and is given at the end of Layer 5.

3.3 Layer 3: Post-Quantum Key Share and ECH

This layer is part of the same `ClientHello` but is worth isolating, because it discriminates between clients that pass classical TLS fingerprinting. The standardised hybrid group `X25519MLKEM768`, which combines classical X25519 with ML-KEM-768 from NIST FIPS 203 [5], shipped by default in Chrome 131 in November 2024 and now appears in the `ClientHello` of every current

Chrome [6].[1] A client that claims to be current Chrome but omits the post-quantum key share contradicts its own user-agent, so an impersonation profile that is merely a year out of date fails here while passing Layer 2. A related signal is Encrypted Client Hello (ECH), which encrypts the server-name field of the handshake using Hybrid Public Key Encryption [7] and which current Chrome uses opportunistically where the server supports it; a client that never offers ECH stands out for the same reason a missing key share does. This too is an *emitted* artifact: adding the correct key share or ECH support to the profile neutralises the check. Its function is to penalise stale profiles rather than to offer structural protection, which is why keeping the impersonation target current matters as much as choosing it correctly.

3.4 Layer 4: HTTP/2 and HTTP/3 Frame Fingerprinting

Above the TLS handshake, the HTTP transport itself is fingerprintable. Browsers open HTTP/2 with a characteristic frame order (`SETTINGS`, then `WINDOW_UPDATE`, then `HEADERS`) and with version-specific `SETTINGS` parameter values. Setting `http2=True` in `httpx` corrects the protocol version but not these parameter values or their order, so a naive HTTP/2 client is still distinguishable from a browser. Current Chrome also prefers HTTP/3 over QUIC [8, 9] where the server advertises it, so a scraper speaking only HTTP/2 to an HTTP/3-enabled zone presents a mild mismatch; QUIC transports carry their own fingerprintable parameters [10]. Like the TLS layers, this reads an *emitted* artifact: the frame sequence and parameter values are chosen by the client and carry no secret, so reproducing a browser's exact transport behaviour defeats the check.

3.5 Layer 5: Header-Order Fingerprinting

The final transport-level check reads the order of request headers rather than their values. Browser engines emit headers in a deterministic, version-specific sequence, and under HTTP/2 the pseudo-headers (`:method`, `:path`, `:scheme`, `:authority`) precede the regular headers in a browser-specific arrangement. A client with a correct `User-Agent` but an anomalous header order is flagged even when every individual header value is correct. This is an *emitted* artifact in its purest form: the ordering is a property of how the client serialises the request, with no secret involved.

A single circumvention for Layers 2 to 5. Although the four layers above read different parts of the request, they are correlated: one faithful-impersonation client reproduces all of them at once, which is why a defender gains little by stacking them (Section 2.2). The practical tool is `curl_cffi` [11], a Python binding over `curl-impersonate` [12] that reproduces a target browser's TLS signature, post-quantum key share, HTTP/2 frame order, and header order in a single call. Recent releases add HTTP/3 fingerprints and the ability to refresh fingerprints at runtime. The interface is close to a drop-in replacement for `requests`:

```python
from curl_cffi import requests
# The bare "chrome" alias auto-tracks the
# latest supported browser fingerprint.
r = requests.get(url, impersonate="chrome")
```

One caveat is itself an instance of Eq. (1). Pinning an outdated profile such as `chrome120` in 2026 becomes a detection signal in its own right, both because no real user runs a two-year-old browser and because the profile lacks the now-common post-quantum key share of Layer 3. The version-tracking alias avoids shipping a stale artifact. For the same reason, passing a custom `headers` dictionary overrides the order that impersonation would otherwise set and defeats the purpose: the order, not only the values, is read.

3.6 Layer 6: Browser and JavaScript Fingerprinting

When a challenge page or a protected page loads, injected JavaScript collects device signals actively. These include canvas and WebGL renderer hashes (headless Chrome reporting "SwiftShader" or "Mesa" is a clear indicator), AudioContext output, font enumeration, the `navigator.webdriver` flag (set to `true` by stock Playwright, Selenium, and Puppeteer), Permissions-API anomalies, plugin lists, screen geometry, and JS-engine timing [13]. This check reads an emitted artifact, but a high-dimensional and tightly coupled one, which raises its effective difficulty: a convincing surface must stay internally consistent across dozens of probes.

A separate hazard for proxy-based scraping is the WebRTC IP leak. STUN requests can expose the host's real address even when HTTP traffic is routed through a proxy. The fix is explicit: pass `--disable-webrtc` to Chromium-based tools, or set `media.peerconnection.enabled = false` for Firefox-based tools such as Camoufox. Current open-source options at this layer are Camoufox [14], a stealth build of Firefox; Patchright [15], a patched, drop-in replacement for the Playwright library that drives Chromium; and, for lighter targets, SeleniumBase in its undetected-Chromedriver mode. The `playwright-extra` stealth plugin, once a common choice, is no longer actively maintained and is now detected reliably.

3.7 Layer 7: CDP and DevTools-Protocol Detection

The Chrome DevTools Protocol (CDP) is the control channel that Playwright, Puppeteer, and Selenium use to drive a browser. Detectors look for CDP artifacts that survive surface-level patches such as setting `navigator.webdriver` to `false`: property de-

[1] Post-quantum key agreement was first enabled experimentally in Chrome 124 (April 2024) using the earlier `X25519Kyber768Draft00` group; the standardised `X25519MLKEM768` replaced it from Chrome 131.

scriptors that CDP modifies at a low level, timing anomalies in instrumented APIs, and internal markers such as `__playwright_target__`. This is the first layer that sits between the two categories. It reads an emitted artifact in principle, but the artifact is the automation channel itself rather than a one-time connection byte sequence, so in practice it behaves partly like a possessed property. The most effective open-source response is `nodriver` [16], the successor to `undetected-chromedriver`, which drives a real Chrome instance over its own DevTools interface rather than through the standard CDP automation path that detectors target. Chromium-based tools such as Patchright patch the best-known CDP leaks, for instance by avoiding `Runtime.enable` and running scripts in isolated execution contexts, but they still operate within Playwright's CDP-based architecture, so they remain exposed to detection methods that target that channel directly. Camoufox takes a different route: it drives Firefox through the Juggler protocol rather than CDP, and isolates Playwright's page agent so that the injected automation scripts are not visible to the page; the residual risk for a Firefox build is engine-level, since SpiderMonkey behaviour cannot be made to look like V8. The trade-off for `nodriver` is that it does not synthesise mouse, scroll, or keystroke dynamics, so it clears CDP detection but must be combined with the behavioural mitigations of Section 3.8.

3.8 Layer 8: Behavioural and Machine-Learning Signals

Since 2025 Cloudflare has deployed per-zone machine-learning anomaly models, trained separately for each protected site, so that behaviour which is normal on a documentation site may be anomalous on a checkout flow [17]. The signals include the regularity of request timing, navigation patterns and referrer chains, scroll and mouse activity, cookie and session age, inter-request intervals, the depth of session history, and the number of concurrent sessions per IP. Cross-request correlation, active since mid-2024, ties fingerprints and behaviour together across a session window, so that a request which looks unremarkable in isolation can stand out over time.

This is a genuine possess-reading layer: the property under inspection is accumulated, per-zone, and non-portable, and it cannot be produced on demand. Effective mitigation therefore tries to accumulate the property rather than fake it. In practice this means non-uniform inter-request delays, drawn from a gamma or exponential distribution rather than a fixed interval; a warm-up that visits the homepage before navigating to deep pages; realistic `Referer` chains; persistent cookies and reused browser profiles, with one profile directory per proxy IP; randomised mouse paths for browser-based tools; and a cap of one to three concurrent sessions per IP. Because the models are per-zone, a configuration that works on one target may fail on another, and per-target calibration is unavoidable for high-value work.

3.9 Layer 9: Managed JavaScript Challenge

When the composite score is merely uncertain rather than clearly bad, Cloudflare returns an interactive challenge instead of a block. The managed challenge is an interstitial that runs JavaScript, evaluates the resulting fingerprint, and on success issues a `cf_clearance` cookie, whose default lifetime is 30 minutes and is configurable by the site operator through the Challenge Passage setting, alongside the per-session `__cf_bm` cookie. The detail that matters most is that `cf_clearance` is bound to the combination of IP address, `User-Agent`, and TLS fingerprint used when it was issued. This binding is what places the layer between the categories: the challenge response is emitted, but the cookie is only useful while the client continues to hold the address it was issued to, which is closer to a possessed property. It also couples the challenge to the IP and behavioural layers, so an adversary must defeat them together. The operational consequence is concrete: a rotating proxy that changes IP between the solve step and the request step invalidates the cookie, so sticky sessions, which keep one IP for the duration of a session, are required. The dominant production pattern is to solve once and reuse: a full stealth browser solves the challenge on a sticky IP, the resulting cookies and the exact `User-Agent` are extracted, and the remaining requests run through `curl_cffi` on the same IP until the cookie expires.

3.10 Layer 10: Turnstile

Turnstile is Cloudflare's replacement for CAPTCHA [18]. It runs a client-side authenticity check before showing any visual puzzle and is often invisible on low-risk sessions, which makes it less an explicit test than a second managed-challenge variant with a widget. Like the managed challenge, it issues a token that is bound to the issuing context, so the same solve-once-and-reuse pattern applies. The two options are a stealth-browser checkbox click, which is free but less reliable, and a paid solving service, which is more reliable; tokens remain valid for roughly five minutes, so a high-volume client must either re-solve on that cadence or fall back to a solving service.

3.11 Layer 11: Bot Fight Mode

The remaining enforcement layers are configuration tiers an operator enables rather than distinct fingerprinting techniques, but each presents a different barrier and deserves separate treatment. Bot Fight Mode, available on the free plan, is the weakest: it is heuristic only, acting on coarse signals such as datacenter origin and obviously non-browser clients. It reads *emitted* artifacts and yields to the baseline impersonation stack of Layers 2 to 5 combined with a residential or mobile address, so a correctly configured `curl_cffi` client clears it.

3.12 Layer 12: Super Bot Fight Mode

On the Pro and Business plans, Super Bot Fight Mode adds a "definitely automated" classification and stricter header checks on top of the free tier. It remains an *emit*-reading layer, but it inspects more of the request, so a client must additionally present correct `Sec-Fetch-*` metadata and a fully consistent header set. The barrier is higher than Bot Fight Mode but the same in kind: a faithful impersonation profile passes.

3.13 Layer 13: Under Attack Mode

Under Attack Mode, available on all plans as a manual toggle, is qualitatively different from the two tiers above. Rather than scoring, it serves a JavaScript challenge to every visitor regardless of reputation, so no request passes on transport fingerprint alone. Because the gate is the challenge of Layer 9, defeating it requires the same solve-once-and-reuse pattern: a stealth browser obtains `cf_clearance` on a sticky IP, after which `curl_cffi` reuses the cookie per request. The check is therefore *emit*-reading but bound to the issuing context, inheriting the IP-binding property of the managed challenge.

3.14 Layer 14: Bot Management

Bot Management is the paid enterprise product and the hardest layer short of cryptographic identity. It is a per-zone machine-learning composite that combines all the prior layers, including the behavioural signals of Layer 8, into a single score trained on the specific site. It reads both *emitted* and *possessed* signals, which is why no single technique defeats it: reliable retrieval requires the whole stack working in concert, and because the model is tuned per zone, a configuration that works against one deployment may fail against another. Results are correspondingly inconsistent, and a heavily tuned zone is the case where a managed scraping service becomes the rational fallback.

3.15 Layer 15: Workers Custom Detection

The most open-ended layer is not a Cloudflare product at all but the operator's own code. Workers custom detection lets a site run arbitrary JavaScript at the edge, implementing honeypot endpoints, custom rate limits, or a bespoke challenge-and-response of its own design. Its emit/possess character is therefore undefined in general: it depends entirely on what the operator chooses to check. By definition no generic bypass exists; each case needs its own analysis, and in practice an unprotected API subdomain (Section 5) is often a more productive avenue than reverse-engineering bespoke edge logic.

3.16 Layer 16: AI Bot Blocker

Two layers target AI crawlers specifically. The AI Bot Blocker, introduced in July 2024, is a one-click toggle that blocks labelled AI user-agents such as `GPTBot` and `CCBot` [19]. It reads a single *emitted* declaration, the `User-Agent` string, and acts on it directly. Because it keys on declared identity rather than behaviour, it is trivially avoided: any client presenting an ordinary browser user-agent passes, which is why this layer constrains only crawlers that honestly identify themselves.

3.17 Layer 17: AI Labyrinth

AI Labyrinth, introduced in March 2025 [20], is subtler and more dangerous, and it is the one layer that does not fit the emit/possess axis cleanly, because it is a behavioural trap rather than a check on the request. Rather than block a suspected bot, it inserts hidden `nofollow` links into a page; these lead to a maze of pre-generated, AI-written decoy pages, built with Workers AI, stored in R2, and screened for cross-site scripting. The links are invisible to humans and marked so that compliant crawlers ignore them, but a scraper that follows every link walks into the maze, where two harms occur at once. The first is data poisoning: the scraper's database fills with plausible but irrelevant content, which, if used for training, can degrade a model. The second is silent fingerprinting: a session that follows the decoy links deep enough is treated as a bot and flagged across Cloudflare's network. What makes this awkward to defend against is the absence of feedback. Unlike a CAPTCHA or a `403`, the labyrinth returns no error, so a scraper can appear to be working while collecting noise and acquiring a network-wide flag. Avoidance is therefore behavioural rather than technical: do not follow `nofollow` or CSS-hidden links; build URL lists from rendered, visible content rather than from raw HTML enumeration; check that retrieved pages remain on topic; and, above all, keep the session in good standing, since the honeypot is served only to sessions already considered suspect.

3.18 Layer 18: Web Bot Auth and Cryptographic Agent Identity

Web Bot Auth, proposed by Cloudflare in May 2025 and folded shortly afterwards into its Verified Bots programme [2, 21], is the first layer in the sequence that reads a possessed property at the protocol level. An agent signs selected components of each request, at minimum the `@authority` it addresses together with a bounded `created` and `expires` validity window, using a private Ed25519 key under the IETF HTTP Message Signatures standard, RFC 9421 [22]. The matching public key is published at a well-known directory, `/.well-known/http-message-signatures-directory`, which the edge retrieves to verify the signature. No amount of TLS, header, or behavioural emulation substitutes for the secret, because the artifact being checked is a signature over the request rather than a reproducible client byte sequence. There is no way to forge it.

Two qualifications are important for an accurate picture. First, the current deployment fails open. A valid signature is a positive identification that exempts the request from bot challenges, but an unsigned request

is not thereby blocked; it simply falls back to the rest of the stack. Web Bot Auth becomes a true admission gate only on a site that explicitly requires a signature, and most do not. Second, as of mid-2026 the specification is an active IETF Internet-Draft [23] rather than a ratified standard. It builds on the already-ratified RFC 9421 but is not itself an RFC. The direction of travel nonetheless points toward wider enforcement: support now extends beyond Cloudflare to AWS WAF, Vercel, Shopify, and Akamai; the payment networks have built on it, with Visa's Trusted Agent Protocol and Mastercard Agent Pay [24]; and several major agent operators, including OpenAI Operator, Browserbase, and Amazon Bedrock AgentCore, sign by default [2]. Where a signature is required, there is no bypass; the only legitimate route is to register as a verified agent.

3.19 Layer 19: Cloudflare Access

Cloudflare Access gates content behind an identity provider, using SSO, OAuth, or an email one-time password. This is authentication rather than anti-bot protection, and it is outside the scope of this paper: circumventing it without credentials would be unauthorised access. Where content sits behind a login, the only legitimate route is authenticated retrieval with a properly obtained account.

4 Summary of Layers

Table 1 brings the nineteen layers together under the emit/possess framework. A clear pattern emerges, with a few honest exceptions. Layers that read a purely emitted artifact, the transport-and-header layers (2 to 5) and the lighter enforcement tiers (11, 12, 16), are reliably bypassable with current tooling, because the artifact can be reproduced. IP reputation (Layer 1) is bypassable too, but economically rather than technically, since the address is emitted while its reputation must be rented. The layers that mix an emitted response with a context binding, the managed challenge, Turnstile, and Under Attack Mode (9, 10, 13), are bypassable only by solving once and reusing on a fixed address, which couples them to the IP layer. Browser and JavaScript fingerprinting (Layer 6) reads an emitted but high-dimensional and tightly coupled artifact, so it is mostly but not trivially bypassable. The two layers that read a partly or wholly possessed property, CDP detection and per-zone behavioural modelling (Layers 7 and 8), are only partly solved, and the enterprise composite (Layer 14) that combines them is correspondingly inconsistent. Workers custom detection (15) and AI Labyrinth (17) sit outside the axis, the first because it is arbitrary operator code and the second because it is a behavioural trap rather than a request check. Only the two layers that rest on a possessed secret, the mandated cryptographic signature of Web Bot Auth and the authentication of Cloudflare Access (18 and 19), admit no technical bypass at all.

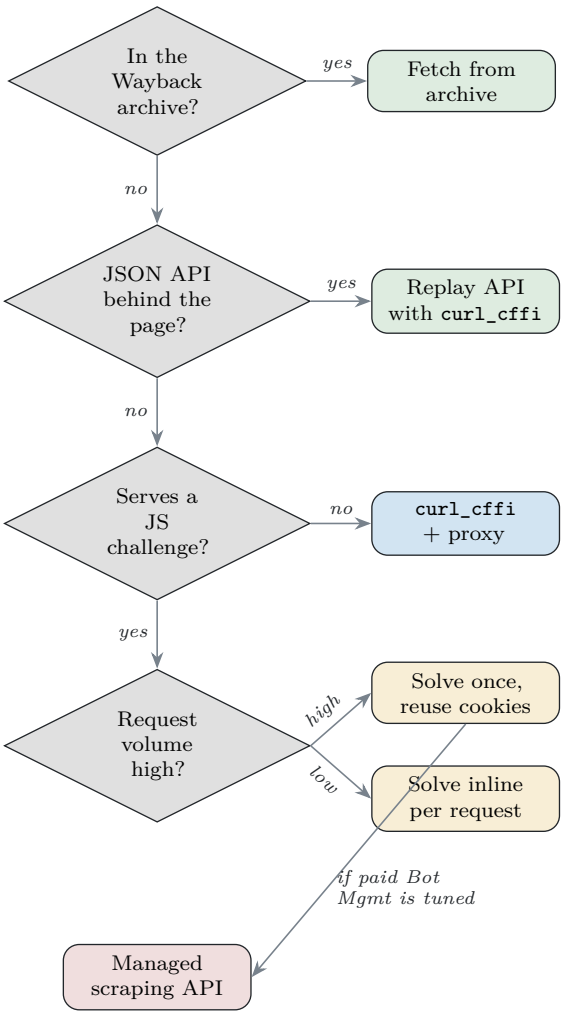

Figure 4: Choosing an approach for a Cloudflare-protected target. The procedure is ordered by cost: an archived snapshot or an unprotected API avoids the live protection stack entirely; `curl_cffi` handles the transport-and-header layers; a stealth browser is reserved for genuine JavaScript challenges; and a managed scraping service is the fallback for a heavily tuned paid Bot Management zone. Terminals are shaded by cost, from cheapest (green) to most expensive (red).

5 Reference Patterns by Scenario

Operational practice settles into a small number of patterns. The order in which to try them follows from the analysis above: the cheapest options come first, and browser automation is a last resort rather than a default. Figure 4 captures this order as a decision procedure.

5.1 Check the Archive First

The cheapest way past a protected site is often not to touch it. The Internet Archive's Wayback Machine serves archived snapshots from `web.archive.org` with no live anti-bot stack, usually as static HTML, and with a historical dimension that the live site cannot offer [25]. The drawbacks are staleness and incompleteness, and `web.archive.org` itself rate-limits. The CDX API lists

Table 1: Consolidated view of the detection mechanisms under the emit/possess framework. Each row is one layer, defined and described in its own subsection of Section 3 under the same number. The final column reflects the state of open-source tooling in mid-2026.

Layer	Mechanism	Evasion type	Difficulty	Best approach	Bypassable?
1	IP reputation	Hybrid	Medium	Sticky mobile-carrier proxy	Yes (economic)
2	TLS / JA3 / JA4	Emit	Medium	`curl_cffi` (impersonate)	Yes
3	Post-quantum TLS + ECH	Emit	Low–Med	`curl_cffi` (auto-included)	Yes (keep current)
4	HTTP/2 + HTTP/3 frames	Emit	Medium	`curl_cffi` (impersonate)	Yes
5	HTTP header order	Emit	Medium	`curl_cffi` (impersonate)	Yes
6	Browser JS + WebRTC	Emit (coupled)	High	Camoufox / Patchright; disable WebRTC	Mostly
7	CDP detection	Emit → Possess	High	`nodriver`	Partially
8	Behavioural / ML (per-zone)	Possess	High	Profile persistence; warming; low concurrency	Partially
9	Managed challenge	Emit + IP-bound	Medium	`nodriver` + cookie reuse (sticky IP)	Yes
10	Turnstile	Emit + bound	Med–High	Stealth click or solving service	Yes (paid)
11	Bot Fight Mode (free)	Emit	Low	`curl_cffi` + residential/mobile IP	Yes
12	Super Bot Fight Mode	Emit	Low–Med	`curl_cffi` + correct `Sec-Fetch-*`	Yes
13	Under Attack Mode	Emit + bound	Medium	`nodriver` + cookie reuse	Yes (per request)
14	Bot Management (paid ML)	Emit + Possess	Very High	Full stack combined	Inconsistent
15	Workers custom detection	Arbitrary	Variable	Per-site analysis	Site-specific
16	AI Bot Blocker	Declaration	Very Low	Browser `User-Agent`	Trivially
17	AI Labyrinth (honeypot)	Behavioural trap	Low if careful	Ignore `nofollow`/hidden; crawl shallow	Avoid by not triggering
18	Web Bot Auth	Possess (crypto)	N/A	none	Fails open now; **No** if mandated
19	Cloudflare Access	Authentication	N/A	none	No (credentials required)

what is archived, and the `id_` suffix returns the raw capture without the Wayback toolbar, giving clean HTML to parse:

```python
import requests

def snapshots(url):
    p = {
        "url": url,
        "output": "json",
        "collapse": "digest",
    }
    cdx = "http://web.archive.org/cdx/search/cdx"
    r = requests.get(cdx, params=p)
    return r.json()[1:]

def raw(ts, url):
    # id_ = capture without toolbar
    a = f"https://web.archive.org/web/{ts}id_/{url}"
    r = requests.get(a, timeout=30)
    return r.text
```

5.2 Intercept the Underlying API

Many applications fetch their data from a JSON API that is separate from the HTML layer, often on a subdomain with weaker or absent Cloudflare coverage, and sometimes shared with a mobile app whose protection is lighter still. Capturing the call in DevTools, filtered to `Fetch/XHR`, and replaying it with `curl_cffi` often removes the need for browser automation altogether, since an API endpoint has no HTML page into which a managed challenge can inject JavaScript.

5.3 The Baseline HTTP Stack

For the majority of Cloudflare-protected targets, which serve no JavaScript challenge, the baseline stack is enough: `curl_cffi` with `impersonate="chrome"` over a residential or mobile proxy, with the headers left to the impersonation profile.

```python
from curl_cffi import requests

s = requests.Session()
r = s.get(
    url,
    impersonate="chrome",
    proxies={"https": "http://user:pass@residential:
     port"}
)
print(r.status_code, len(r.text))
```

5.4 Solve Once, Reuse Many Times

For challenged targets at volume, solve the challenge once with a stealth browser on a sticky IP, then reuse the cookies through `curl_cffi` on the *same* IP. The two steps must share one address, since `cf_clearance` is bound to it:

```python
import nodriver as uc
from curl_cffi import requests as cffi

STICKY = "http://user:pass@sticky:port"

async def solve(url):
    # stealth browser, sticky IP
    b = await uc.start(browser_args=[
        f"--proxy-server={STICKY}",
        "--disable-webrtc",
    ])
    p = await b.get(url)
    await p.sleep(4)
    ck = await p.browser.cookies([url])
    b.stop()
    return {c["name"]: c["value"] for c in ck}

def scrape(url, ck):
    # lightweight reuse, same IP
    s = cffi.Session()
    s.cookies.update(ck)
    r = s.get(url,
              impersonate="chrome",
              proxies={"https": STICKY})
    return r.text
```

5.5 High-Security Targets Running Paid Bot Management

Against a zone with Cloudflare's paid Bot Management actively tuned, maintaining a bespoke bypass becomes a continuous engineering cost, and a managed scraping API such as ScrapFly, Bright Data, ScrapingBee, or ZenRows is frequently cheaper per request than the engineering time it would save. A minimum do-it-yourself stack combines all of the following: a sticky mobile-carrier IP; `curl_cffi` with the version-tracking alias; `cf_clearance` and `__cf_bm` obtained from `nodriver` with a persistent profile; WebRTC disabled; a session warm-up; gamma-distributed delays; the same proxy IP for the solve and for every fetch; and a cap of one to three concurrent sessions per IP.

5.6 Error Handling

Reliable operation depends on reading status codes correctly. A `200` response may still carry a challenge body, recognisable from markers such as `__cf_chl_` or the phrase "checking your browser", and calls for a re-solve. A `403` or `1020` indicates a hard block and calls for IP rotation. A `429` or `1015` indicates rate limiting and calls for exponential backoff. A `503` indicates Under Attack Mode. A `1010` indicates a browser-integrity failure at the CDP layer, which calls for `nodriver` or Camoufox.

6 Techniques That No Longer Work

Several once-standard techniques are now detected reliably or are structurally inadequate, and each failure traces back to a specific layer in Section 3. `cloudscraper` has fallen behind repeated changes to the challenge format. Older FlareSolverr configurations are detected by their browser fingerprint, and although the tool is still maintained, its effectiveness against current challenges is inconsistent and depends heavily on the proxy and browser engine behind it. The `playwright-extra` stealth plugin is no longer actively maintained. Standalone `undetected-chromedriver` does not address CDP detection. Datacenter proxies sit in blocklisted ASNs. Raw `requests` and `httpx` present the wrong TLS signature, HTTP/2 frames, and header order. Hard-pinned impersonation profiles lack the post-quantum key share and are anomalous in their own right. And rotating proxies break `cf_clearance` reuse, because the cookie is bound to the issuing IP, `User-Agent`, and TLS fingerprint.

7 Legal and Ethical Considerations

The ability to circumvent a technical control is not a legal entitlement to do so, and several considerations bound legitimate practice. Most sites prohibit automated access in their terms of service, and a violation may carry civil liability in some jurisdictions even for public content. Computer-misuse statutes, including the CFAA in the United States and the Computer Misuse Act in the United Kingdom, may attach risk where access circumvents a protection measure, although the United States position on public data was substantially narrowed by *hiQ Labs v. LinkedIn* [26]. Even permitted scraping can become tortious if the request volume materially impairs a server's ability to serve other users. Where scraped data includes personal information, regimes such as the GDPR and the CCPA govern its storage and processing. And while `robots.txt` is not generally binding, respecting it is an established norm that is weighed in assessments of intent.

The techniques described here are documented for the legitimate collection of public data, such as price monitoring, research, accessibility tooling, archival, and competitive intelligence. Where an official API exists, it should be preferred, because it is faster, more stable, and free of legal ambiguity.

8 Conclusion

Modern bot protection is a composite of many layers, and Eq. (1) captures the asymmetry at its centre: an adversary must raise the minimum across all active layers, whereas a defender need add only one layer that is close to impassable. The emit/possess distinction then indicates where durable resistance comes from. The transport-fingerprint layers attract the most attention from practitioners, yet they are the shallowest, because they read emitted artifacts that a faithful imitator can reproduce, and because they behave as a single term in the bound. Durable resistance comes instead from the layers that read a possessed property: per-zone behavioural history, which is non-portable and must genuinely be accumulated, and cryptographic agent identity, which cannot be forged where a signature is required.

The same analysis yields a practical order of operations. Check the archive first; intercept an API next;

reach for `curl_cffi` to handle the transport-and-header layers together; turn to `nodriver` only when a genuine JavaScript challenge requires it; and treat the two layers that admit no technical bypass, a mandated signature and authentication, as limits to respect rather than obstacles to defeat. As impersonation tooling continues to close the gap on emitted artifacts, the centre of gravity in bot management is shifting toward properties that cannot be emitted on demand. AI Labyrinth, which poisons rather than blocks, and Web Bot Auth, which replaces imitation with a held secret, are early instances of that shift.

Acknowledgements

The time-sensitive claims in this paper were independently re-verified against primary sources current to June 2026, including the Cloudflare engineering blog, the IETF datatracker, and the documentation of the tools cited.

References

[1] Cloudflare. *Cloudflare Radar: Adoption & Usage Trends*. https://radar.cloudflare.com/. Accessed June 2026. 2026.

[2] T. Meunier. *Forget IPs: Using Cryptography to Verify Bot and Agent Traffic*. https://blog.cloudflare.com/web-bot-auth/. 2025.

[3] J. Althouse, J. Atkinson, and J. Atkins. "TLS Fingerprinting with JA3 and JA3S". In: *Salesforce Engineering Blog* (2017).

[4] J. Althouse. *JA4+: Network Fingerprinting Standard*. https://github.com/FoxIO-LLC/ja4. FoxIO, LLC. 2023.

[5] National Institute of Standards and Technology. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. FIPS. U.S. Department of Commerce, 2024.

[6] Google Chrome Security Team. *A Year of Post-Quantum Cryptography on Chrome*. https://security.googleblog.com/. Standardised X25519MLKEM768 shipped by default in Chrome 131, Nov. 2024. 2024.

[7] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. *RFC 9180: Hybrid Public Key Encryption*. RFC. IETF, 2022.

[8] J. Iyengar and M. Thomson. *RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC. IETF, 2021.

[9] M. Bishop. *RFC 9114: HTTP/3*. RFC. IETF, 2022.

[10] FoxIO. *JA4T and QUIC Fingerprinting*. https://github.com/FoxIO-LLC/ja4. 2024.

[11] Y. Tang. *curl_cffi: Python Binding for curl-impersonate*. https://github.com/lexiforest/curl_cffi. 2026.

[12] D. Sleren et al. *curl-impersonate: A Special Build of curl that Impersonates Browsers*. https://github.com/lexiforest/curl-impersonate. 2026.

[13] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine. "Browser Fingerprinting: A Survey". In: *ACM Transactions on the Web* 14.2 (2020), 8:1–8:33. DOI: 10.1145/3386040.

[14] daijro. *Camoufox: Anti-Detect Firefox for Web Scraping*. https://github.com/daijro/camoufox. 2026.

[15] Kaliiiiiiiiii-Vinyzu. *Patchright: Undetected Python Version of the Playwright Library*. https://github.com/Kaliiiiiiiiii-Vinyzu/patchright-python. 2026.

[16] UltrafunkAmsterdam. *nodriver: Successor to undetected-chromedriver*. https://github.com/ultrafunkamsterdam/nodriver. 2026.

[17] Cloudflare. *Bot Management and Bot Score Reference*. https://developers.cloudflare.com/bots/. Accessed June 2026. 2026.

[18] Cloudflare. *Cloudflare Turnstile: A CAPTCHA Alternative*. https://blog.cloudflare.com/turnstile-private-captcha-alternative/. 2022.

[19] Cloudflare. *Declare Your AIndependence: Block AI Bots, Scrapers and Crawlers with a Single Click*. https://blog.cloudflare.com/declaring-your-aindependence-block-ai-bots-scrapers-and-crawlers-with-a-single-click/. 2024.

[20] Cloudflare. *Trapping Misbehaving Bots in an AI Labyrinth*. https://blog.cloudflare.com/ai-labyrinth/. Announced March 19, 2025. 2025.

[21] Cloudflare. *Message Signatures are Now Part of Our Verified Bots Program*. https://blog.cloudflare.com/verified-bots-with-cryptography/. 2025.

[22] A. Backman, J. Richer, and M. Sporny. *RFC 9421: HTTP Message Signatures*. RFC. IETF, 2024.

[23] T. Meunier et al. *Web Bot Auth Architecture (Internet-Draft draft-meunier-web-bot-auth-architecture)*. https://datatracker.ietf.org/doc/draft-meunier-web-bot-auth-architecture/. IETF Internet-Draft, work in progress. 2025.

[24] Visa. *Visa Trusted Agent Protocol (TAP)*. https://corporate.visa.com/. 2025.

[25] Internet Archive. *Wayback CDX Server API*. https://github.com/internetarchive/wayback/tree/master/wayback-cdx-server. 2026.

[26] United States Court of Appeals, Ninth Circuit. *hiQ Labs, Inc. v. LinkedIn Corp., No. 17-16783*. 9th Cir. 2022.