When SNI Goes Dark: How Encrypted Client Hello Changes Network Security

August 13, 2026

An SOC analyst pulls a firewall log after an alert from their EDR flags a suspicious process reaching out to the internet. The expectation is routine: find the SNI field in the logged TLS handshake, check it against a blocklist, confirm or dismiss. Instead, the destination field reads something like `cloudflare-ech.com`, a generic front door shared by tens of thousands of unrelated sites. The firewall did its job. The handshake completed. Nothing was blocked. And the analyst hasn't a clue where the process actually went.

The gap stems from the Encrypted Client Hello (ECH), now a finished IETF standard as of March 2026 (Rescorla et al., 2026 (https://www.rfc-editor.org/rfc/rfc9849.html)). Most coverage of ECH treats it as a privacy upgrade: one more plaintext leak closed, one more win for privacy-first users. That framing is correct as far as it goes. It also skips over the part that matters to anyone in a SOC: a huge amount of deployed security tooling was built on the assumption that the destination hostname would keep showing up in cleartext, and ECH revokes that assumption for every connection where it succeeds.

SNI Was Never Meant to Be a Security Control

Server Name Indication was standardized in 2011 to solve a hosting problem, not a security one (Eastlake, 2011 (https://datatracker.ietf.org/doc/html/rfc6066)). Before SNI, a TLS server needed a dedicated IP address per certificate, because the client had no way to say which hostname it wanted before the server picked a certificate to present. SNI let the client announce the hostname in the clear, in the very first message of the handshake, so one IP address could serve certificates for thousands of independent sites. A game changer for virtual hosting.

Network security vendors adopted it for a different reason, and the timing explains why. Under TLS 1.2, a firewall performing URL-category filtering or malware detection on encrypted traffic could read the server's certificate in cleartext during the handshake and derive a hostname from it. TLS 1.3 encrypted the certificate. SNI was the field left standing, so inspection logic shifted to it across the board: middleboxes started treating a field designed for certificate selection as an authoritative signal for policy enforcement (Campling et al., 2026 (https://datatracker.ietf.org/doc/html/draft-campling-ech-deployment-considerations-12)). The IETF noticed this dependency forming years before ECH shipped. As early as 2020 a catalogue began documenting exactly the kind of middlebox behavior SNI had accumulated, from enterprise firewalls blocking inappropriate websites to firewalls exempting healthcare and financial domains from inspection, stating clearly that encrypting the field would "break or reduce" the efficacy of the operational practices and techniques implemented in middleboxes (Huitema, 2020 (https://www.rfc-editor.org/rfc/rfc8744)).

To be fair, defenders never fully trusted SNI. It can be omitted, spoofed, or set to a value that doesn't match the certificate the server ultimately presents, so mature deployments treat it as a provisional signal rather than ground truth (Campling et al., 2026 (https://datatracker.ietf.org/doc/html/draft-campling-ech-deployment-considerations-12)). But that provisional, cross-checked status is still a load-bearing role. Take the field away and the cross-check has nothing to check.

Two Hellos, One Blind Spot

How ECH works at a conceptual level is rather straightforward. The client builds two versions of the `ClientHello` message. The outer one goes out in the clear and carries a generic "public name": a cover identity shared by every site behind the same front-end, plus an `encrypted_client_hello` extension holding the real message, sealed with a public key. The inner one is the real `ClientHello`: the actual SNI, ALPN list, cipher preferences, unreadable to anyone but the server. The server decrypts the inner message after receiving the outer one and proceeds as if the inner message were the only one ever sent (Rescorla et al., 2026 (https://www.rfc-editor.org/rfc/rfc9849.html)).

But that public key has to come from somewhere the client can reach prior to the handshake actually beginning, so it comes from DNS: a client resolves the target's HTTPS resource record, and the ECH configuration, including the HPKE public key and public name, travels inside it. Two consequences follow directly. First, ECH is only as trustworthy as the DNS path that delivers it, which is why real-world ECH deployments lean on encrypted DNS to reach the resolver in the first place. Second, an on-path party that can strip or spoof that DNS record can force a client back to a plaintext handshake before ECH ever begins: a detail that matters a great deal for defenders and attackers alike.

This is where the design gets openly adversarial toward middleboxes. RFC 9849 requires TLS 1.3 clients to send a GREASE ECH extension, a syntactically valid but non-functional `encrypted_client_hello`, even on connections where the client has no real ECH configuration for the target (Rescorla et al., 2026 (https://www.rfc-editor.org/rfc/rfc9849.html)). The extension format is identical in both cases; a passive observer has no way to distinguish a GREASE payload from a genuine HPKE-sealed `ClientHelloInner` without the server's private key. Blocking on the mere presence of the extension means blocking nearly all TLS 1.3 traffic, since the extension is close to universal regardless of whether real encryption is happening underneath.[1] For the destination specifically, the outer handshake reveals almost nothing: everything else in it — cipher suites, extensions, ALPN — stays legible, and stays that way for the fingerprinting techniques covered below. That asymmetry is not an oversight; it's the point.

The Controls That Just Lost Their Anchor

URL category and reputation filtering read the SNI, make a query against a database, and allow or block before any payload is inspected. Selective TLS decryption policies use the SNI to decide which sessions get the expensive MITM treatment, so that a firewall doesn't have to intercept every connection to a bank

or healthcare provider along with every connection to suspicious networks. Threat-intelligence blocklists, the kind fed by indicators of compromise from a threat feed, match against that same field. DLP and Cloud Access Security Broker (CASB) tooling also often key off destination hostnames as a first-pass filter before deeper content inspection kicks in. None of these mechanisms are exotic; they are what "next-generation firewall" has meant in practice for the better half of the last decade.

Two sectors feel this more than general enterprise IT. Education providers with statutory content-filtering needs have leaned on SNI-based filtering specifically because it works without decrypting payloads on devices the school doesn't own, something evermore common with BYOD student devices. Regulated finance is the other case: firms are often required to keep records of inbound and outbound communications as an insider-trading control, and an audit trail with SNI-shaped holes in it is a compliance problem, not just a visibility one.

Vendors aren't hiding these facts. Palo Alto Networks' own documentation recommends blocking the DNS record types that carry ECH configuration, on grounds that letting ECH through can interfere with firewall services that depend on reading the `ClientHello` (Palo Alto Networks, 2026 (https://docs.paloaltonetworks.com/pan-os/11-0/pan-os-new-features/content-inspection-features/dns-security-support-for-dns-over-https)). That's not a hypothetical concern. It's a major firewall vendor telling its own customers that a core capability degrades by default the moment ECH succeeds.

The Attacker's Side of the Ledger

It would be nice if this were purely a privacy-versus-visibility trade-off with no offensive angle. It isn't quite that clean, but the attacker upside is narrower than it might first appear.

A good example of this is ECHidna, a RAT that researchers at NTT Security Holdings identified in early 2024 in targeted attacks against Japanese organizations. ECHidna routes its command-and-control traffic through ECH, so its C2 connections are structurally identical, at the network layer, to ordinary browser traffic reaching Cloudflare (Sawabe & Koike, 2025 (https://www.virusbulletin.com/uploads/pdf/conference/vb2025/papers/Stealth-over-TLS-the-emergence-of-ECH-based-C-C-in-ECHidna-malware.pdf)). Any control keyed to domain reputation, URL category, or SNI-based blocking has nothing to inspect: the SNI it would inspect isn't there. ECH isn't the malware's only evasion layer, either. ECHidna also calls low-level Windows socket functions directly, rather than a standard HTTP library, and builds its command structures in raw heap memory instead of a parseable format, aimed at the host-based detection ECH doesn't touch (Sawabe & Koike, 2025 (https://www.virusbulletin.com/uploads/pdf/conference/vb2025/papers/Stealth-over-TLS-the-emergence-of-ECH-based-C-C-in-ECHidna-malware.pdf)). Between the two layers, network-based and host-based detection are each missing a different half of the same connection.

Set against that, ECH turns out to be a weaker censorship-circumvention tool than its early advocates hoped. A 2025 measurement study testing ECH handshakes against censorship infrastructure in China, Iran, and Russia found the practical benefit limited: ECH support among TLS servers is concentrated almost entirely in Cloudflare, and a meaningful share of Cloudflare's own servers don't advertise it, so a client attempting ECH frequently finds no usable ECH configuration to negotiate against. Where it is

negotiable, Russian DPI was found actively detecting and blocking the ECH extension itself, and China and Iran were found censoring the encrypted-DNS lookups ECH depends on rather than fighting the encrypted handshake directly (Niere et al., 2025 (https://www.petsymposium.org/foci/2025/foci-2025-0016.pdf)). The finding, in short: ECH doesn't remove the censor's leverage so much as relocate it one layer down, to DNS.

The same logic applies inside the enterprise. ECH hides where a connection is going. It does nothing to hide that a connection happened, roughly how big it was, when it occurred, or what the client's TLS stack looks like structurally: exactly the fields that survive in the outer, unencrypted `ClientHello`. The destination is hidden. The shape of the conversation is not.

What the Wire Still Tells You

That surviving shape is where post-ECH detection has to live: the signals below are independently actionable, not degraded stand-ins for the one that disappeared.

TLS fingerprinting has matured specifically to be resilient against this collapse of many destinations into one shared identity. The older JA3 method hashed a `ClientHello`'s cipher suites and extensions in the exact order the client sent them, which made it fragile by construction: when Chrome began randomizing that extension order in 2023, specifically to stop middleboxes from hardcoding assumptions about `ClientHello` structure rather than to defeat fingerprinting, a single Chrome installation started producing a different JA3 hash on nearly every connection, and JA3-based detection built around it degraded within weeks (Fastly, 2023 (https://www.fastly.com/blog/a-first-look-at-chromes-tls-clienthello-permutation-in-the-wild)). Its successor, JA4, sorts both the cipher list and the extension list into a fixed order before hashing, which cancels out exactly that kind of reordering, and produces a structured, human-readable fingerprint such as `t13d1516h2_8daaf6152771_e5627906d626` rather than an opaque hash (FoxIO, 2024 (https://github.com/FoxIO-LLC/ja4)). Every one of the inputs to that fingerprint, including TLS version, cipher list, extension set, and ALPN, lives in the outer `ClientHello`, the part ECH leaves untouched. A connection can hide its destination and still hand a defender a stable, comparable signature of the client software making it. A stock Chrome install looks different from a Go HTTP client, which looks different from a C2 loader, independent of where any of them are going.

IP address and ASN reputation survive intact; ECH is not a VPN and does nothing to the network layer. Traffic volume and timing survive too: a passive observer still sees how many bytes moved and when, and behavioral analysis built on that shape, the same approach public network operators were already forced toward, doesn't care what the SNI said. DNS telemetry survives as well, in exactly the cases where an organization controls its own resolution path: if every endpoint is forced through a resolver the organization operates, that resolver sees the lookup that fetches the ECH configuration in the first place, even though it can't see the TLS handshake that follows.

There's one more path that deserves a name, because it doesn't route around ECH so much as ignore it entirely: a forward TLS proxy that terminates the client's connection and opens a fresh one to the origin never has to deal with the client's ECH at all, because it is the effective destination as far as the client's

TLS stack is concerned. This is precisely how corporate MITM decryption already works, and it is why it remains the single most complete answer to ECH's visibility problem, with a cost attached that the next section takes seriously.

Finally, and often the most durable signal of all: process-level context from the endpoint. A firewall that can correlate a network connection with the binary that opened it, the kind of integration modern EDR-to-NGFW pipelines increasingly provide, doesn't need the SNI to notice that a bare `wget` invocation or an unsigned binary is making ECH-protected connections to a CDN at 3 a.m., while every legitimate connection from that host is coming from a recognizable browser process (Cisco Systems, 2025 (https://secure.cisco.com/secure-firewall/v7.6/docs/encrypted-client-hello-defense-strategies-how-cisco-secure-firewall-tackles-ech)). ECH blinds the network layer. It has nothing to say about the endpoint.

Replacing a Signal Without Rebuilding Surveillance

The blunt fix, decrypting everything all the time, was already unpopular before ECH, for good reason: it puts the organization in the business of running its own CA, trains users to click through certificate warnings, and creates a single point of failure that is itself an attractive target. ECH makes that trade-off more tempting; it doesn't make it any less worth resisting. A more proportionate response layers narrower tools, each aimed at a specific gap rather than at all traffic indiscriminately.

Start with DNS, since it's the one dependency ECH cannot avoid: a client that never receives the ECH configuration falls back to an ordinary, cleartext-SNI handshake by design. Forcing all endpoint DNS traffic through an internal, organization-controlled resolver, and blocking outbound DNS-over-HTTPS, DNS-over-TLS, and DNS-over-QUIC to anything else, is the same control NSA guidance has recommended for encrypted-DNS visibility generally, not something invented for ECH specifically (National Security Agency, 2021 (https://media.defense.gov/2021/Jan/14/2002564889/-1/-1/0/CSI_ADOPTING_ENCRYPTED_DNS_U_OO_102904_21.PDF); Cybersecurity and Infrastructure Security Agency, 2024 (https://www.cisa.gov/sites/default/files/2024-05/Encrypted%20DNS%20Implementation%20Guidance_508c.pdf)). The internal resolver can then be configured to strip the `ech=` parameter from HTTPS records before answering, which is a materially different move from disabling encryption outright: it controls which party gets to see the destination, rather than removing DNS confidentiality on the wire between the endpoint and that resolver. This is where visibility and privacy stop being purely opposed: the organization's own resolver, not an arbitrary third party, ends up positioned to see what it's obligated to see.

On managed endpoints, browser policy can disable ECH and non-approved encrypted DNS directly, and this remains one of the more complete mitigations available today. It's also the least durable one. Policy support for disabling ECH lives at vendor discretion, non-browser applications built on TLS libraries with ECH baked in frequently expose no policy hook at all, and BYOD or guest devices are outside the reach of any endpoint policy by definition (Campling et al., 2026 (https://datatracker.ietf.org/doc/html/draft-campling-ech-deployment-considerations-12)). Treat this layer as a control that narrows the problem for a known population of devices today, not a permanent architectural guarantee.

For traffic that gets through anyway, selective decrypt-and-resign aimed narrowly at known ECH-capable CDN destinations is a materially different posture than blanket interception. The mechanism runs in two legs, not one. First, an inline firewall completes the client-facing handshake itself, presenting a certificate re-signed by an internal CA the endpoint already trusts; that satisfies the one condition RFC 9849 sets before a client will treat ECH as declined and safely retry in cleartext, rather than treat the interference as an attack. Second, the firewall opens its own, separate connection to the real ECH front-end with the extension stripped, so the front-end proceeds as an ordinary non-ECH handshake on that leg. Once the client retries with real SNI in cleartext, exactly as the protocol's own "securely disabled" logic instructs it to, the firewall intercepts that retry too and finally sees the actual destination (Cisco Systems, 2025 (https://secure.cisco.com/secure-firewall/v7.6/docs/encrypted-client-hello-defense-strategies-how-cisco-secure-firewall-tackles-ech)).[2] Combined with process-level or JA4-based triage to flag non-browser clients using ECH, this gets a defender most of the way back to their original visibility for the traffic that matters most, without re-decrypting the entire organization's browsing history to do it.

None of this is free, and the IETF draft that catalogs these operational impacts is explicit that any visibility-restoring mitigation has to be weighed against the new attack surface, misuse potential, and data-protection obligations it introduces in its own right (Campling et al., 2026 (https://datatracker.ietf.org/doc/html/draft-campling-ech-deployment-considerations-12)).[3] The honest way to frame the trade-off is this: ECH forces a choice between accepting a real loss of network-layer visibility and reaching for controls such as DNS steering, selective decryption, and endpoint correlation, all of which carry cost and risk of their own. There is no configuration change that makes that choice disappear.

Key Takeaways

1. ECH became a finished IETF standard, RFC 9849, in March 2026. It encrypts the entire `ClientHello`, including SNI, using a key bootstrapped from a DNS HTTPS record.
2. Security tooling built to key off cleartext SNI (URL filtering, selective decryption, DLP/CASB domain checks, statutory content filtering, compliance audit trails) degrades or stops working outright once ECH succeeds.
3. TLS 1.3 clients are required to send a GREASE ECH extension, a non-functional placeholder in the same format as real ECH, on ordinary connections, so a firewall cannot reliably tell real ECH traffic from traffic that never used it, which rules out simply blocking the extension.
4. Real malware has already used ECH for C2 concealment; ECH's benefit for state-level censorship evasion has been measured as more limited, since censors increasingly target the encrypted-DNS bootstrap instead.
5. IP/ASN reputation, traffic timing and volume, JA4-style TLS fingerprints, DNS telemetry at a controlled resolver, and endpoint process context all survive ECH intact: collectively a workable, if less convenient, substitute for hostname-based detection.
6. The most durable defensive posture layers DNS control, endpoint policy where enforceable, and narrowly targeted selective decryption, rather than reverting to blanket TLS interception.

References

Campling, A., Vixie, P., Wright, D., Taddei, A., & Edwards, S. (2026). Encrypted Client Hello Deployment Considerations (Internet-Draft, individual submission, no IETF consensus status). IETF. https://datatracker.ietf.org/doc/html/draft-campling-ech-deployment-considerations-12

Cisco Systems. (2025). Encrypted Client Hello Defense Strategies: How Cisco Secure Firewall Tackles ECH. https://secure.cisco.com/secure-firewall/v7.6/docs/encrypted-client-hello-defense-strategies-how-cisco-secure-firewall-tackles-ech

Cybersecurity and Infrastructure Security Agency. (2024). Encrypted DNS Implementation Guidance. https://www.cisa.gov/sites/default/files/2024-05/Encrypted%20DNS%20Implementation%20Guidance_508c.pdf

Eastlake, D. (2011). Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, IETF. https://datatracker.ietf.org/doc/html/rfc6066

Farrell, S., & Tschofenig, H. (2014). Pervasive Monitoring Is an Attack. BCP 188 / RFC 7258, IETF. https://www.rfc-editor.org/rfc/rfc7258

Fastly. (2023). A First Look at Chrome's TLS ClientHello Permutation in the Wild. https://www.fastly.com/blog/a-first-look-at-chromes-tls-clienthello-permutation-in-the-wild

FoxIO. (2024). JA4+: A Suite of Network Fingerprinting Standards. GitHub. https://github.com/FoxIO-LLC/ja4

Huitema, C. (2020). Issues and Requirements for Server Name Identification (SNI) Encryption in TLS. RFC 8744, IETF. https://www.rfc-editor.org/rfc/rfc8744

National Security Agency. (2021). Adopting Encrypted DNS in Enterprise Environments. https://media.defense.gov/2021/Jan/14/2002564889/-1/-1/0/CSI_ADOPTING_ENCRYPTED_DNS_U_OO_102904_21.PDF

Niere, N., et al. (2025). Encrypted Client Hello (ECH) in Censorship Circumvention. Free and Open Communications on the Internet (FOCI) 2025. https://www.petsymposium.org/foci/2025/foci-2025-0016.pdf

Palo Alto Networks. (2026). DNS Security Support for DNS Over HTTPS (DoH). PAN-OS New Features Guide. https://docs.paloaltonetworks.com/pan-os/11-0/pan-os-new-features/content-inspection-features/dns-security-support-for-dns-over-https

Rescorla, E., Oku, K., Sullivan, N., & Wood, C. A. (2026). TLS Encrypted Client Hello. RFC 9849, IETF. https://www.rfc-editor.org/rfc/rfc9849.html

Sawabe, Y., & Koike, R. (2025). Stealth over TLS: The Emergence of ECH-Based C&C in ECHidna Malware. Virus Bulletin Conference 2025. https://www.virusbulletin.com/uploads/pdf/conference/vb2025/papers/Stealth-over-TLS-the-emergence-of-ECH-based-C-C-in-ECHidna-malware.pdf

Footnotes

1. The anti-discrimination property runs deeper than the extension's mere presence. The `ServerHello` message encodes, in the last eight bytes of its random field, a cryptographic signal of whether ECH negotiation actually succeeded — readable only by a party holding the matching HPKE keys. A middlebox observing the handshake cannot distinguish those bytes from ordinary randomness, so it can't even infer success or failure after the fact, let alone in real time. ↩

2. This depends entirely on the client trusting the certificate the firewall re-signs for the client-facing leg, which means it works only on managed endpoints where an internal CA has already been deployed to the trust store. It has no effect on guest networks or unmanaged devices, and vendor guidance separately warns that simply blocking connections to ECH-capable CDN endpoints outright, rather than working through this decline-and-retry sequence, produces inconsistent browser retry behavior and multi-second connection delays for users. ↩
3. It's worth naming the tension directly, and it's textual, not inferred: the same short BCP that supplies ECH's ethical grounding, the assessment that pervasive monitoring is itself an attack on privacy, also states plainly that "[m]aking networks unmanageable to mitigate PM is not an acceptable outcome" (Farrell & Tschofenig, 2014). Neither constraint has ever formally overridden the other; the document leaves the balance to be worked out case by case. ↩
