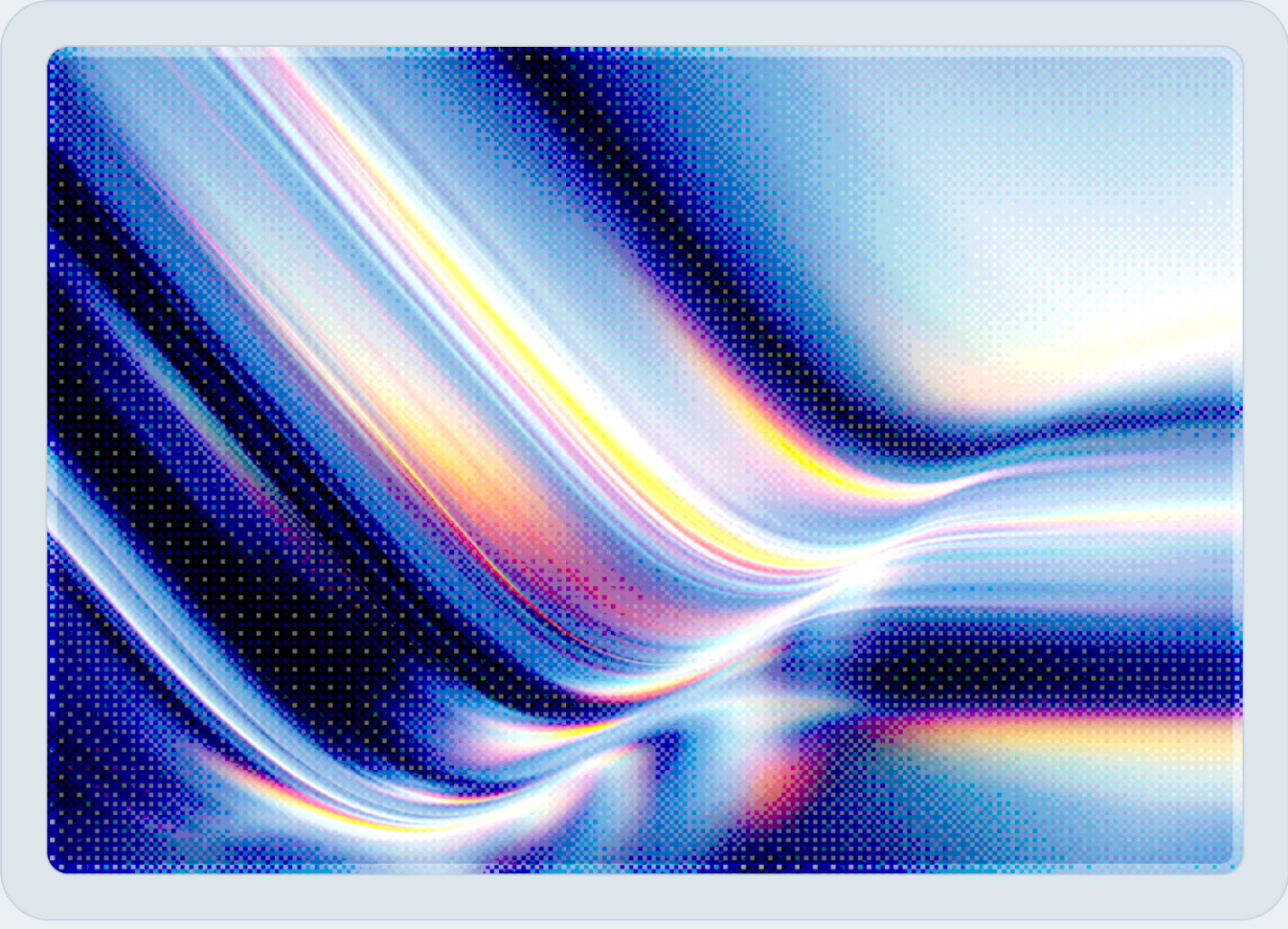
CSAI Foundation | Cloud Security Alliance

OPTIMJACK: Compromised Optimizely Scripts and Client-Side Supply Chain Risk

How one trusted experimentation snippet delivered a wallet swapper, card skimmer, and iOS exploit loader to 178 organizations

2026-11-18

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

- On November 16, 2026, an attacker wrote malicious loader code into the customer-editable Project JavaScript field of Optimizely Web Experimentation projects. Optimizely's own CDN then served that code, with a valid certificate, to visitors of 178 organizations, including the BBC, CNN, Zoom, Salesforce, GitLab, and HSBC [1][2][3].
- The payload swapped cryptocurrency wallet addresses, skimmed card numbers from first-party checkout forms, sent cookies and screenshots to attacker servers, and on iPhones and iPads loaded the Coruna and DarkSword exploit kits [1][2]. Public scans show exploit pages being delivered. No device takeover had been confirmed as of November 17 [2][4].
- Optimizely attributes initial access to a vulnerability in its Experimentation account signup flow [3]. It has not explained how a signup-flow flaw resulted in malicious code across many unrelated customer projects, which suggests, but does not confirm, cross-tenant write access [2]. No CVE has been assigned, as is common for flaws in hosted SaaS services [4].
- Allowlisting a vendor domain gave no protection, because the malicious code came from the trusted host. Of 159 organizations whose Content Security Policies (CSPs) Report URI classified, only 24 (15%) would have blocked the payload. Common directives such as `'strict-dynamic'` and `'unsafe-eval'` weakened otherwise restrictive policies [1].
- Organizations that load any Optimizely Web snippet should audit Project JavaScript and experiment code now. They should treat card data and non-HttpOnly session cookies captured on affected pages between roughly 04:00 UTC on November 16 and at least 23:11 UTC (when Report URI reported all infected snippets clean) as exposed, allowing additional time for cached copies of the snippet [1]. They should also add third-party script governance to their vendor risk programs.

Background

Optimizely Web Experimentation is a widely used A/B testing and personalization platform. Customers embed a one-line JavaScript snippet on their pages, and the browser loads a project-specific file, usually from `cdn.optimizely.com/js/<project ID>.js`. That file contains Optimizely's static runtime code along with dynamic, customer-authored content: experiment variations, audience logic, and an optional "Project JavaScript" block that runs on every page where the snippet loads. The architecture

exists so that marketing and product teams can change page behavior without an engineering release. That same design means anyone with write access to a project can execute arbitrary JavaScript in the origin of every site that embeds it. Because the snippet usually loads early and synchronously so that experiments render without flicker, it typically runs with the full privileges of the hosting page.

Abuse of this class of dependency is not new. In June 2024, Sansec reported that the `polyfill.io` domain, sold to a new owner earlier that year, was injecting redirect malware into more than 100,000 websites that still loaded its script [8]. In late July 2026, Adform's `trackpoint-async.js` tracking script was altered to swap cryptocurrency addresses on customer sites, and Adform reported detecting the suspicious activity and removing the code on July 27 [9]. CSA's August 2026 research note on that week's incidents described the pattern as concentration risk, in which a single shared script, CDN, or registry becomes a single point of failure for thousands of downstream parties [10]. OPTIMJACK differs from those cases mainly in its targets and its payload. It reached some of the most heavily trafficked properties on the web and combined theft with mobile exploit delivery.

The campaign name "OPTIMJACK" comes from independent researcher reporting rather than from Optimizely. Scott Helme and Report URI published the most detailed primary analysis, and StepSecurity, Ostorlab, and BleepingComputer published corroborating timelines [1][2][3][4]. Report URI and Ostorlab both report that an earlier wave between November 13 and 16 injected a `webshell[.]lol` loader through Kissmetrics, StorifyMe, and Minute Media scripts [1][4]. If that holds, the Optimizely compromise was one stage of a broader campaign against third-party script providers rather than an isolated event. Because that linkage traces back to the same small group of researchers and has not been independently confirmed, it should be treated as provisional.

Security Analysis

Injection Vector and Timeline

According to Report URI, the attacker modified only the `projectJS` field of affected projects. The rest of each snippet was untouched, and the content came from Optimizely's origin, not from CDN cache poisoning [1]. Sites that proxied the snippet through their own domain, or self-hosted copies that synchronized from Optimizely during the exposure window, were also affected, because the file content still came from the compromised project [4]. Optimizely told BleepingComputer that an unauthorized party "exploited a vulnerability in the Experimentation account signup flow" [3]. The company says it fixed the flaw, removed the injected code, ended active sessions, revoked tokens, reset access for affected accounts, blocked code delivery at its Cloudflare edge, and notified affected customers [2][3].

Optimizely's own `www.optimizely.com` and `login.optimizely.com` also carried the loader [1][2]. This suggests the attacker could write to projects across tenant boundaries, not just to a single compromised customer account. That is an inference from the observed breadth, not a confirmed root cause.

Counts of the affected population differ by source and by unit of measurement, and readers comparing figures should keep the distinction in mind. Report URI's sweep between 19:24 and 19:59 UTC found 186 infected snippets among 1,611 scanned, a figure later revised to 188 snippets belonging to 178 organizations [1][3]. Report URI separately crawled 256 of the affected organizations' own hosts and classified the CSPs of 159 organizations [1]. Because one organization may operate several snippets and hosts, these numbers measure overlapping but different populations rather than conflicting totals.

Helme identified three loader variants with different evasion properties [1]. Most snippets used a plain script tag pointing to `1688360[.]cc/track.js`. A second variant hid the same URL in base64 and decoded it with `atob()`, which defeats a simple string search for the domain. A third variant embedded the full payload inline with an `eval` fallback, so blocking the payload domain at the network layer would not stop execution. During the day the attacker rebuilt the payload 26 times, adding capabilities and wallets without removing any [1][4]. Because each build had a different hash, file-hash indicators match only the specific build they were taken from.

The table below reconstructs the incident timeline from the primary sources. Note that the first observed injection, at 04:21 UTC, loaded the earlier-wave `webshell[.]lol` injector rather than `1688360[.]cc`, which explains why it precedes the payload domain's registration by about an hour [1][2].

Time (UTC)	Event
Nov 16, 04:21	First Optimizely injection observed, in a BBC snippet (loading `webshell[.]lol`) [1][2]
Nov 16, 05:23	Payload domain `1688360[.]cc` registered [2]
Nov 16, 08:10–08:28	First `track.js` build (wallet clipper); live on bbc.com [1][4]
Nov 16, 13:43	Card skimmer module added [1]
Nov 16, 17:13–17:57	Second iOS exploit host added; Coruna iframe injected for iOS 18.6 and earlier [4]

Time (UTC)	Event
Nov 16, 19:24–19:59	Report URI sweep finds 186 of 1,611 snippets infected [1]
Nov 16, 20:40–21:10	Mass cleanup of infected snippets [1][2]
Nov 16, 21:52	One of 188 infected snippets still serving the payload [1][2]
Nov 16, 23:11	Report URI reports all infected snippets clean [1]
Nov 17, 00:37	`1688360[.]cc` placed on registry clientHold [1][2]
Nov 17, 04:11–05:39	Attacker empties several cryptocurrency wallets [2][4]

Payload Capabilities

Each module of the payload exploited the hosting page's trust in a different way. The wallet-address swapper held replacement addresses for at least eight chains: ETH, TRX, BTC, TON, SOL, LTC, BCH, and DOGE. To limit false positives, it replaced only addresses that passed the chain's checksum and appeared near payment-related text. It hooked page text, form inputs, copy events, the clipboard API, `fetch`, and XHR, so an address copied from an affected page could already be swapped by the time it was pasted into a wallet [2]. Ostorlab additionally reports interception of links, images, navigation, and the iOS share sheet [4]. After a swap, the script captured the visible page with `html2canvas` and posted the screenshot, the page URL, non-HttpOnly cookies, and the original addresses to `/hit.php` [1][2].

The card skimmer, added mid-afternoon UTC, wrapped `fetch` and XHR and listened for form submissions. It looked for Luhn-valid card numbers with known brand prefixes, valid expiry dates, and CVVs, then sent them to `/card.php` using `navigator.sendBeacon` [1][2]. It could not reach payment fields rendered in cross-origin iframes, such as hosted fields from payment processors [1][4], a limitation that explains why payment architecture appears among the recommendations below.

The iOS module checked for iPhone and iPad user agents and loaded `boot.js` from a rotating set of `.icu` hosts. The sources describe the kits' targeting in two inconsistent ways. Helme's version table attributes iOS 13–17 to the Coruna kit and iOS 18.x to the DarkSword kit, while the incident timelines published by both Report URI and Ostorlab describe a Coruna iframe gated to iOS 18.6 and earlier [1][4].

Under either reading, devices on iOS 18.7.3 or later fell outside the reported exploit targeting, and Helme reported that iOS 26 devices were only fingerprinted [1]. StepSecurity did not retrieve the second-stage files and notes that public scans show exploit pages being delivered, not successful exploitation [2].

Directly traceable financial losses appear modest compared with the exposure. Report URI counted about $7,040 received by attacker wallets as of 19:25 UTC on November 16, and Ostorlab later cited about $7,150 [1][4]. Some of the wallets were active before the attack, so these figures may overstate theft attributable to OPTIMJACK [3]. The number of card numbers, cookies, and screenshots exfiltrated is unknown [4]. These probably represent the larger risk, since session cookies can enable account takeover and screenshots of logged-in pages can contain personal data. A low cryptocurrency total should therefore not be read as evidence of low impact.

Why Existing Controls Underperformed

OPTIMJACK is instructive because it bypassed the control most organizations rely on for third-party scripts: trust in the vendor's domain. Report URI's classification of 159 organizations' CSPs shows how little protection deployed policies offered against a payload served from an allowlisted host [1].

CSP outcome	Organizations	Share
No CSP	68	43%
Policy allowed both payload loading and data theft	50	31%
Policy blocked data exfiltration only	17	11%
Policy blocked the payload outright	24	15%
Total classified	**159**	**100%**

Having `cdn.optimizely.com` in the allowlist gave no protection, since that host served the loader. Two widely deployed settings weakened protection further. `'strict-dynamic'` extends trust to any script inserted by an already trusted script, so it effectively passed trust from the Optimizely snippet to the attacker's loader. `'unsafe-eval'`, which Report URI notes Optimizely's documentation recommends, allowed the inline fallback to run. At HSBC, the inline fallback executed through `eval` even though the policy blocked `track.js` itself [1].

A restrictive `connect-src` directive was effective against the card, cookie, and screenshot modules, because each needed a network request to an attacker host. It did nothing against the wallet swapper, which modifies content entirely inside the page [1]. Subresource Integrity (SRI), the standard browser mechanism for pinning a script to a known hash, is hard to apply to experimentation snippets, because their content changes legitimately every time a marketer edits an experiment [5]. Optimizely documents a Cloudflare Worker integrity check for the snippet, but that check hashes only the static core and explicitly leaves dynamic experiment configuration to internal governance [7]. Based on that documentation, the check appears unlikely to have flagged a modification confined to Project JavaScript. This illustrates a general pattern: integrity tooling covers the parts of a third-party script that rarely change, while attackers target the parts that change by design.

Helme argues that organizations scrutinize first-party server code through review, testing, and approval, yet let third-party JavaScript change silently in production. As a result, most cannot say what code their pages executed on a given day [5]. PCI DSS v4.0.1 Requirements 6.4.3 and 11.6.1 address this gap for payment pages through script inventory, authorization, and tamper detection [6], and those requirements have been mandatory since March 31, 2025 [14]. However, they cover only payment pages. OPTIMJACK ran site-wide, and on affected properties without first-party card forms, such as news sites, the card skimmer had nothing to capture while the cookie, screenshot, and wallet modules still operated.

Recommendations

Immediate Actions

Any organization that loads an Optimizely Web Experimentation snippet, whether from `cdn.optimizely.com` or a self-hosted path, should review the Project JavaScript field and all experiment custom code in every project, including inherited and dormant projects, and remove anything the team did not author. Searching for `1688360` alone is insufficient, because the base64 and inline variants do not contain that string in plain text. Reviewers should also look for `atob(`, `eval`, `track.js`, `webshell.lol`, and the global variables `window._k7`, `window._cc`, `window._qxc`, and `window._shot` [1][2][4]. Optimizely user accounts, API tokens, and integrations with write access should be reviewed and rotated, and customers should formally ask Optimizely which project IDs were modified, when, and through which account [2][4]. Because Optimizely cleaned files during the evening of November 16, a clean snippet today does not prove it was clean during the exposure window.

Security teams should block the published indicators in DNS and proxy controls and search web, proxy, and CSP-report logs back to at least 04:00 UTC on November 16, the earliest observed injection [2]. Organizations that also load Kissmetrics, StorifyMe, or Minute Media scripts should extend that search to November 13 to cover the earlier wave [1][4]. The indicators include `1688360[.]cc`, `webshell[.]lol`, `ios-hack[.]icu`, and the randomized `.icu` hosts listed by StepSecurity and Ostorlab. Organizations whose own forms collected card data on affected pages during the window should treat those cards as exposed and work with their acquirers on notification. Session tokens stored in non-HttpOnly cookies should be invalidated and rotated [1][4].

Customer-facing and device-level follow-up completes the immediate response. Where customers may have copied cryptocurrency addresses from affected pages, organizations should consider advising them to verify recent transfers and to confirm destination addresses on a hardware-wallet display. Mobile device management teams should identify managed iPhones and iPads that visited affected sites during the window and confirm that they run patched iOS releases. Report URI recommends iOS 18.7.3 or later, or 26.3 or later [1]; the 26.3 floor presumably reflects exposure of earlier iOS 26 releases to the underlying vulnerabilities, even though iOS 26 devices were only fingerprinted in this campaign.

Short-Term Mitigations

CSP is, in our assessment, the most widely deployable browser-side control for this attack class, but OPTIMJACK shows that policies need to be tuned for third-party compromise rather than only for cross-site scripting. Organizations should restrict `connect-src` to the specific endpoints their pages need, because that directive would have stopped this payload's card, cookie, and screenshot exfiltration [1]. It should be paired with restrictive `img-src`, `form-action`, and `script-src` directives, since an attacker who adapts can move exfiltration to image requests, form submissions, or other channels that `connect-src` does not govern. Organizations should also remove `'unsafe-eval'` wherever the vendor integration can tolerate it and reassess `'strict-dynamic'` on pages that load vendor-managed scripts. Teams that cannot move to an enforcing policy immediately should deploy CSP in Report-Only mode with reporting enabled, which provides near-real-time visibility into new script sources and exfiltration destinations without breaking production [5].

Payment and credential pages warrant a stricter posture than the rest of a site. Card entry should move into a payment processor's cross-origin hosted fields or iframe, which kept card data out of this skimmer's reach [1][2]. Hosted fields reduce but do not eliminate risk, however, because script running in the parent page can still overlay or replace the payment iframe with a counterfeit form. For that reason, experimentation, analytics, and advertising tags should be removed from checkout, login, and account-recovery flows entirely, unless a documented business justification exists and is reviewed. Where Optimizely or a similar tool must remain, organizations should consider using the vendor's self-hosting or

integrity options as a partial control, while recognizing that these typically do not cover dynamic, customer-authored code [7]. Internal change governance over who can publish Project JavaScript should be at least as strict as governance over first-party code deployment.

Strategic Considerations

The broader lesson is that third-party client-side code is a production software supply chain and should be governed like one. Organizations should keep a continuously updated inventory of every external script that executes on their properties, with an owner, a business justification, and a record of what code actually ran over time. Tag managers and experimentation platforms deserve particular attention because they are designed to execute arbitrary code without an engineering release. The PCI DSS 6.4.3 and 11.6.1 model of inventory, authorization, and tamper detection is a reasonable template to extend beyond payment pages to every page that handles authentication, personal data, or financial transactions [6].

Vendor risk programs should also change how they assess providers of embedded scripts. Due diligence should cover the provider's tenant isolation for write access to customer projects, its change logging and customer-visible audit trails, its incident notification commitments, and whether its recommended integration requires weakening the customer's CSP. OPTIMJACK, Adform, and Polyfill.io together show a recurring pattern of a single script provider becoming a cross-industry single point of failure [8][9][10]. Organizations should therefore plan for the next provider compromise as a resilience problem: map which vendors can execute code on which pages, and keep a tested procedure for disabling any one of them quickly. Salesforce did this when it disabled Optimizely scripts across preview and production instances as a precaution [3].

CSA Resource Alignment

CSA's research note *Single Points of Failure: A Week of Supply Chain Compromises* (August 2026) analyzed the most closely related prior incident, the Adform tracking-script compromise, which used the same wallet-swapping technique [10]. That note examined Adform alongside contemporaneous registry and CDN incidents and argued that concentration in shared services turns a single vendor compromise into an ecosystem-wide event. OPTIMJACK is consistent with that analysis and extends it, operating at greater scale and adding card skimming, session theft, and mobile exploit delivery to the same delivery channel. Organizations that adopted that note's resilience and blast-radius recommendations, particularly mapping which shared services can execute code in their environment, would likely have been able to scope exposure more quickly on November 16.

npm Supply Chain Under Siege: TeamPCP, Miasma, and npm v12 (June 2026) examines JavaScript supply chain compromise on the build side, covering credential-stealing worms, CI/CD pipeline abuse, and registry-level structural responses [11]. OPTIMJACK is the runtime counterpart, in which code that never passes through a package manager, lockfile, or build pipeline is fetched directly by end-user browsers on every page load. Organizations that have invested in software composition analysis and build provenance for npm dependencies should recognize that those controls do not see third-party scripts loaded at runtime. They should consider extending inventory and change-monitoring practices to the client side.

ShinyHunters' OAuth Pivot: A Year of SaaS Supply-Chain Breaches (July 2026) traces how durable, overlooked third-party trust (in that case, OAuth tokens granted to integrated SaaS vendors) became the path into customer data [12]. The parallel is close. An Optimizely snippet is a standing grant of code execution to a vendor, and that grant is rarely revisited after initial deployment. The governance recommendations in that report, including periodic re-attestation of third-party access and rapid revocation playbooks, apply directly to embedded scripts.

For control mapping, CSA's AI Controls Matrix (AICM) v1.1 provides the relevant structure [13]. The Supply Chain Management, Transparency, and Accountability (STA) domain addresses third-party inventory, risk assessment, and contractual security requirements. The Application and Interface Security (AIS) domain covers application security baselines that should include CSP and client-side integrity. The Threat and Vulnerability Management (TVM) and Logging and Monitoring (LOG) domains support the detection and indicator-sweep activities described above. Assessors using CSA STAR can draw on these domains to ask embedded-script vendors about tenant isolation and customer-visible change auditing.

References

[1] Scott Helme / Report URI. "World's Largest Brands Infected With Malware Via Optimizely." Report URI Blog, November 16, 2026 (updated November 17, 2026).

[2] Owais Shaikh / StepSecurity. "OPTIMJACK: Optimizely-Hosted Scripts Loaded a Crypto Clipper and Card Skimmer on 178 Organizations." StepSecurity Threat Intel, November 17, 2026.

[3] Lawrence Abrams. "Optimizely hack infected CNN, Fox News, BBC, Salesforce sites with malware." BleepingComputer, November 16, 2026.

[4] Ostorlab. "OPTIMJACK: Optimizely Scripts Served Card Skimmer." Ostorlab Blog, November 17, 2026.

[5] Scott Helme. "Reflecting on the Optimizely Incident and Why We Keep Failing." scotthelme.co.uk, November 17, 2026.

[6] Scott Helme / Report URI. "PCI DSS 4.0.1 and Requirements 6.4.3 and 11.6.1: What Changed for Payment Page Scripts." Report URI Blog, August 18, 2026.

[7] Optimizely. "Snippet integrity check with Cloudflare." Optimizely Web Experimentation Developer Documentation, accessed November 18, 2026.

[8] Sansec. "Polyfill supply chain attack hits 100K+ sites." Sansec Research, June 25, 2024.

[9] Bill Toulas. "Online ad firm Adform's script compromised to steal cryptocurrency." BleepingComputer, July 31, 2026.

[10] Cloud Security Alliance AI Safety Initiative. "Single Points of Failure: A Week of Supply Chain Compromises." Cloud Security Alliance, August 5, 2026.

[11] Cloud Security Alliance AI Safety Initiative. "npm Supply Chain Under Siege: TeamPCP, Miasma, and npm v12." Cloud Security Alliance, June 11, 2026.

[12] Cloud Security Alliance AI Safety Initiative. "ShinyHunters' OAuth Pivot: A Year of SaaS Supply-Chain Breaches." Cloud Security Alliance, July 16, 2026.

[13] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, June 22, 2026.

[14] PCI Security Standards Council. "PCI DSS v4.0.1 Document Library." PCI Security Standards Council, accessed November 18, 2026.