Hijacked at the Source: AppsFlyer SDK Crypto Stealer

Domain Registrar Compromise Delivers Wallet-Hijacking JavaScript Across the Mobile Analytics Supply Chain

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-18

Key Takeaways

Between approximately 20:40 UTC on March 9, 2026 and 10:30 UTC on March 10, 2026—a window of roughly fourteen hours—malicious JavaScript was served from `websdk.appsflyer.com`, the official content delivery domain for AppsFlyer's Web SDK. Any website or application loading the SDK during this window potentially exposed its users to a sophisticated crypto-asset theft payload that silently replaced cryptocurrency wallet addresses in browser form inputs, API request and response bodies, and clipboard contents. AppsFlyer attributed the root cause to a "domain registrar incident," a category of infrastructure compromise that bypasses conventional application-layer security controls entirely.

The scale of potential exposure is substantial. AppsFlyer's marketing analytics platform is used by 15,000 businesses [1], including major consumer brands across e-commerce, fintech, and digital entertainment verticals. While no specific organization has been publicly named as a confirmed victim with documented financial loss, the breadth of the SDK's deployment means that any entity loading the AppsFlyer Web SDK during the exposure window should treat the incident as a confirmed supply chain compromise requiring immediate log review and user notification evaluation.

This incident follows an established pattern. Like the Polyfill.io compromise of 2024 and the Magecart skimming campaigns that preceded it, the AppsFlyer attack demonstrates that trusted third-party JavaScript loaded from external CDN domains constitutes an uncontrolled dependency that can be weaponized without any action on the part of the hosting application. The timing of the incident is particularly notable from a regulatory standpoint: PCI DSS 4.0.1 Requirements 6.4.3 and 11.6.1, which mandate script inventory, integrity verification, and tamper detection for payment pages, became mandatory on March 31, 2025 [2]. The AppsFlyer incident illustrates that operationalizing these controls remains a challenge for many organizations, even after PCI DSS 4.0.1's enforcement deadline.

Background

AppsFlyer and the Third-Party SDK Trust Model

AppsFlyer is an Israeli mobile measurement and marketing analytics company whose platform enables app developers and digital marketers to attribute user acquisition, measure campaign performance, and analyze in-app behavior across mobile and web channels. The Web SDK—distinct from the company's mobile SDKs for iOS and Android—is loaded as a JavaScript snippet into web properties, where it fires analytics events and interfaces with AppsFlyer's backend. Because the SDK is served directly from `websdk.appsflyer.com` rather than self-hosted by customers, all applications using it share a single point of failure: the integrity of that domain and its CDN infrastructure [1].

This architecture, sometimes called a "tag manager" or "third-party analytics script" model, trades operational convenience for a meaningful expansion of the application's trust boundary. From the browser's perspective, JavaScript served from `websdk.appsflyer.com` runs in the same origin context as the host page: it can read and modify DOM content, intercept network requests via fetch and XMLHttpRequest overrides, observe clipboard events, and access any data the user types into form fields. The application operator has no practical mechanism to restrict what the loaded script does without implementing explicit controls such as Content Security Policy headers or Subresource Integrity verification—controls that industry data suggests remain inconsistently deployed across many web properties.

Domain Registrar Incidents as a Supply Chain Vector

AppsFlyer described the March 9–10 event as a "domain registrar incident," a phrase that covers a range of scenarios in which an attacker gains unauthorized control over a domain's DNS configuration or certificate management through the registrar or registrar-adjacent infrastructure rather than through the application stack itself. The practical consequence is that the attacker can redirect DNS records, obtain fraudulent TLS certificates, or replace content served from the domain without needing to compromise any of AppsFlyer's application servers, source repositories, or deployment pipelines. Conventional software supply chain security measures—dependency scanning, build attestation, SBOM generation—provide no protection against this class of attack, because the content is not modified in the build phase but at the delivery layer.

The Profero Incident Response Team [1], Rescana [3], and Feroot Security [4] independently confirmed the compromise through payload analysis of archived copies of the malicious script served during the exposure window. Community researchers documented payload details in public forums beginning

approximately March 10, 2026 at 08:12 UTC [5], roughly two hours before the malicious payload was removed.

Security Analysis

Payload Architecture: Stealth Through Functional Preservation

A particularly dangerous characteristic of the injected payload was that it preserved the legitimate SDK's marketing analytics functionality. A developer or security analyst observing SDK behavior during the compromise window would have seen normal attribution and analytics events firing correctly; the malicious layer operated silently underneath. This design significantly reduces the likelihood of detection through behavioral monitoring unless security tooling is specifically watching for address substitution or unexpected network requests to analytics-origin endpoints.

The payload employed layered obfuscation to resist static analysis. String literals were encoded using a base91-like scheme with shuffled alphabets, decoded at runtime through multiple chained functions. Dead code—including LRU cache implementations, linked-list data structures, SHA-256 routines, and tree-balancing algorithms—was interspersed throughout the payload to increase its apparent complexity and make automated analysis more difficult [3]. No plaintext strings indicating malicious intent were visible, meaning the script was unlikely to be flagged by signature-based static analysis tools relying primarily on string pattern matching [3].

Network Interception and Address Replacement

The payload's primary attack mechanism relied on two complementary techniques: network request interception and DOM observation. The injected code replaced the browser's native `fetch` API with a proxy function and patched `XMLHttpRequest` at the prototype level, allowing it to inspect and modify outbound network requests and inbound responses made through the fetch and XMLHttpRequest APIs before they reached either the application or the user. This approach intercepts cryptocurrency addresses not only when users type them into forms, but also when addresses are retrieved from server-side APIs and displayed to users—a capability that significantly broadens the attack surface beyond simple form field skimming.

Complementing the network interceptors, the payload deployed `MutationObserver` instances to monitor DOM changes in real time. As the application dynamically rendered content, the payload scanned newly inserted nodes for cryptocurrency address patterns using a set of format-specific regular

expressions covering five major blockchain ecosystems. Clipboard monitoring was also implemented, so that even addresses copied from elsewhere and pasted into the application would be replaced before submission. The combination of these three interception layers—network, DOM, and clipboard—created a system in which a user conducting a cryptocurrency transaction on an affected web property during the exposure window was likely to have their intended recipient address silently replaced with an attacker-controlled address, particularly for transactions involving the five confirmed cryptocurrency ecosystems.

Command-and-Control via the Compromised Domain

A design decision that reflects attacker sophistication was the use of the compromised `websdk.appsflyer.com` domain itself for command-and-control operations, rather than a separate attacker-controlled server. The payload fetched current attacker wallet addresses from `websdk.appsflyer.com/v1/api/plugin` and submitted exfiltrated data to `websdk.appsflyer.com/v1/api/process` [3]. Exfiltrated data payloads—containing the original intended wallet address, page URL, timestamp, and browser user agent string—were XOR-obfuscated before transmission. This architecture was effective because network monitoring and firewall rules that blocklist suspicious third-party domains would pass all C2 traffic without inspection: from the network's perspective, the application was simply communicating with its known analytics vendor.

Dynamic wallet fetching, in which the payload retrieved attacker addresses from the C2 endpoint at runtime rather than embedding them statically, prevented defenders from neutralizing the attack by blocklisting specific wallet addresses. The attacker could rotate receiving addresses as frequently as desired without redeploying the payload. The confirmed receiving addresses during the March 9–10 window encompassed five major cryptocurrency ecosystems: Ethereum, Bitcoin, Solana, Ripple (XRP), and TRON [3]. The payload's data structure additionally included a field suggesting Monero support, though active Monero address replacement was not confirmed in analyzed samples [3].

Scope of Exposure and Detection Challenges

The fourteen-hour exposure window, spanning European evening hours through Asian business morning hours on March 9–10, 2026 UTC, likely captured active user sessions across global time zones. Because the attack targeted web properties rather than mobile applications—AppsFlyer confirmed the mobile SDK was not affected [6]—the affected population consists of users interacting with web-based applications, web views, and browser-based interfaces for those applications that load the Web SDK. AppsFlyer has stated that no customer data on its own systems was accessed as part of the incident [6], but this statement speaks to the company's server-side environment and does not address data exfiltrated from end users' browsers during the compromise window.

Consumer-facing brands confirmed to use AppsFlyer's platform—and therefore potentially exposed through their web properties—include TikTok, Netflix, Ubisoft, Carrefour, Burger King, Lululemon, and Pepsi [1], though none of these organizations have confirmed active exploitation of their users. The practical difficulty is that detecting whether a specific user's transaction was intercepted requires correlating transaction records with browser session logs timestamped to the March 9–10 UTC window, a forensic exercise that many applications may not be readily instrumented to perform.

Recommendations

Immediate Actions

Organizations that load the AppsFlyer Web SDK in any web property should begin with log review. The primary signal is connections from user browsers to `websdk.appsflyer.com/v1/api/plugin`, `/v1/api/process`, or `/v1/api/process?rd=` during the window of March 9, 2026 20:40 UTC through March 10, 2026 10:30 UTC [3]. These endpoints are distinct from AppsFlyer's legitimate analytics API surface and their appearance in access logs or network monitoring data constitutes a positive indicator of exposure. Organizations should also inventory which of their web properties load the AppsFlyer Web SDK dynamically, as opposed to a self-hosted copy, since only the dynamically loaded version was affected.

Following log review, organizations should evaluate their notification obligations. If user-facing cryptocurrency transaction functionality was available on affected web properties during the exposure window, the risk of actual address substitution is non-trivial, and affected users may have directed cryptocurrency transfers to attacker-controlled addresses. Engagement with legal and privacy counsel to assess notification requirements under applicable data protection regulations and financial services rules is appropriate given this risk profile.

AppsFlyer revoked all API V2 tokens generated before March 10, 2026 at 19:00 UTC and required customers to regenerate tokens [6][7]. Organizations that have not yet completed this rotation should do so immediately, as unrevoked tokens from the pre-compromise period represent a residual credential exposure risk independent of the JavaScript payload.

Short-Term Mitigations

The AppsFlyer incident is a concrete example of why Content Security Policy and Subresource Integrity controls exist. Organizations should audit all third-party JavaScript loaded from external domains on web properties that handle sensitive data—particularly authentication flows, financial transactions, and user account management. For scripts that must be loaded from external CDNs, implementing SRI hashes (the `integrity` attribute on `<script>` tags) ensures that browsers reject substituted content even if the serving domain is compromised. For scripts where SRI is not practical due to frequent updates by the vendor, a strict Content Security Policy limiting which domains may serve JavaScript and restricting the use of `eval()` and inline scripts can meaningfully reduce the attack surface.

Organizations operating payment pages or financial transaction interfaces should treat the PCI DSS 4.0.1 requirements as a template even if they are not directly in scope for PCI compliance. Requirement 6.4.3 mandates maintaining a justified inventory of all scripts loaded on payment pages, implementing integrity controls, and documenting business justification for each [2]. Requirement 11.6.1 mandates automated monitoring for unauthorized modifications to payment page content and headers [2]. Both requirements address exactly the class of attack demonstrated by this incident.

Strategic Considerations

The AppsFlyer incident belongs to a broader pattern of supply chain attacks exploiting trusted third-party JavaScript delivery infrastructure. The Polyfill.io compromise in June 2024, in which a Chinese entity acquired the polyfill.io domain and began serving malware to approximately 100,000 websites [13], demonstrated that domain acquisition and domain registrar manipulation are viable vectors for large-scale web supply chain attacks. Defenders cannot rely on the reputation or track record of a third-party vendor as a sufficient control: the attack surface is the delivery infrastructure, which may change ownership, be misconfigured, or be compromised through registrar-level attacks that are invisible to the application vendor itself.

Organizations should develop a third-party script security program that treats all externally hosted JavaScript as inherently untrusted regardless of vendor reputation. This means conducting periodic audits of all third-party script dependencies, requiring vendors to document their CDN security practices including registrar security (hardware tokens, registrar lock, DNSSEC), and establishing monitoring that alerts on unexpected changes to script content served from third-party domains. For high-risk web properties—those handling payments, authentication, or financial data—evaluating the feasibility of self-hosting critical vendor scripts with a controlled update workflow provides substantially stronger supply chain integrity guarantees.

CSA Resource Alignment

This incident intersects directly with CSA's guidance across several frameworks. The **Cloud Controls Matrix (CCM) v4.0** addresses third-party dependency risk under the Supply Chain Management, Transparency, and Accountability (STA) domain, particularly STA-08 (Supply Chain Risk Management) and STA-09 (Third-Party Assessment), which together require organizations to assess the security posture of vendors providing critical infrastructure components—a class that clearly includes SDK delivery CDNs [8]. The Application Security (AIS) domain's controls on software development and testing (AIS-01 through AIS-04) provide a framework for extending security requirements to software components sourced from third parties [8].

CSA's **AI Organizational Responsibilities** guidance is relevant insofar as AI-driven analytics tools and marketing intelligence platforms increasingly occupy the same third-party script ecosystem that the AppsFlyer attack exploited. As organizations integrate AI-powered analytics, attribution, and behavioral intelligence tools, each integration point represents a potential supply chain vector that requires the same scrutiny applied to any externally hosted JavaScript dependency [9].

The **MAESTRO threat modeling framework** for agentic AI systems offers a useful analytical lens for understanding why this attack pattern is particularly difficult to detect. The conceptual parallels are notable: MAESTRO's analysis of how trust boundaries can be subverted by agents operating within shared execution environments illuminates why the injected payload was so difficult to distinguish from legitimate behavior—even if the framework was developed for AI agent contexts rather than browser-based supply chain attacks [10]. The injected payload observed network traffic, modified data in transit, communicated with C2 infrastructure, and exfiltrated data, all within the trust context granted to the legitimate AppsFlyer SDK.

CSA's **Zero Trust guidance** emphasizes that trust should never be implicit based on network location, identity, or historical behavior. The AppsFlyer incident is an example of implicit trust failure: applications trusted the AppsFlyer CDN domain unconditionally because it had historically delivered legitimate content. A Zero Trust architecture applied to the browser execution environment—through SRI verification where scripts are version-pinned, strict Content Security Policy, and behavioral monitoring of outbound connections—could have detected or significantly limited the impact of the malicious script [11].

The **CSA STAR (Security Trust Assurance and Risk)** program provides a mechanism for customers to assess vendor security posture through the Cloud Controls Matrix. Organizations that use AppsFlyer or comparable third-party analytics vendors should use STAR documentation to evaluate vendor controls over delivery infrastructure security, including CDN and domain registrar security practices, as part of their third-party risk management processes [12].

References

1. Profero IRT, "Hijacked at the Source — AppsFlyer's SDK Distributes a Crypto Stealer," profero.io, March 2026. [https://profero.io/blog/hijacked-at-the-source-a-trusted-marketing-appsflyers-sdk-distributes-a-crypto-stealer]

2. PCI Security Standards Council, "PCI DSS v4.0.1 Requirements 6.4.3 and 11.6.1," pcisecuritystandards.org, 2024. [https://www.pcisecuritystandards.org/document_library/]

3. Rescana, "AppsFlyer Web SDK Supply Chain Attack Analysis," rescana.com, March 2026. [https://www.rescana.com/post/appsflyer-web-sdk-supply-chain-attack-global-crypto-stealing-javascript-injection-and-mitigation-an]

4. SC Media, "AppsFlyer SDK compromise confirmed by Feroot Security," scmagazine.com, March 11, 2026. [Referenced via SC Media reporting; direct URL to be verified]

5. cometkim, "AppsFlyer SDK compromised 2026-03-10," GitHub Gist, March 10, 2026. [https://gist.github.com/cometkim/5bea18688e1653d2c3fe5476d3efed12]

6. AppsFlyer, customer communication regarding domain registrar incident and API token revocation, March 10, 2026. [Referenced via BleepingComputer and Cyberwarzone reporting; direct URL to be verified]

7. BleepingComputer, "AppsFlyer Web SDK hijacked to spread crypto-stealing JavaScript code," bleepingcomputer.com, March 2026. [https://www.bleepingcomputer.com/news/security/appsflyer-web-sdk-used-to-spread-crypto-stealer-javascript-code/]

8. Cloud Security Alliance, "Cloud Controls Matrix v4.0," cloudsecurityalliance.org, 2021. [https://cloudsecurityalliance.org/research/cloud-controls-matrix/]

9. Cloud Security Alliance, "AI Organizational Responsibilities: Core Security Responsibilities," cloudsecurityalliance.org, 2024. [https://cloudsecurityalliance.org/research/ai-safety/; URL pending verification — original link returned 404]

10. Cloud Security Alliance, "MAESTRO: Agentic AI Threat Modeling," cloudsecurityalliance.org, 2025. [https://cloudsecurityalliance.org/research/ai-safety/; URL pending verification — original link returned 404; verify dedicated MAESTRO document page]

11. Cloud Security Alliance, "Zero Trust Advancement Center," cloudsecurityalliance.org. [https://cloudsecurityalliance.org/research/topics/zero-trust/]

12. Cloud Security Alliance, "STAR: Security Trust Assurance and Risk," cloudsecurityalliance.org. [https://cloudsecurityalliance.org/star/]

13. Sansec, "Polyfill supply chain attack hits 100K+ sites," sansec.io, June 25, 2024. [https://sansec.io/research/polyfill-supply-chain-attack]

