SUPPLY-CHAIN INCIDENT ANALYSIS

The Adform Trackpoint Compromise

How a Single Ad-Tech Script Became a Cryptocurrency-Address-Swapping Supply-Chain Attack

A technical and operational analysis for security engineers, ad-operations teams, and third-party risk managers

August 2026 | Compiled from public reporting and researcher disclosures

Contents

Executive Summary

In late July 2026, attackers modified a JavaScript tracking library operated by Adform — a Copenhagen-headquartered, European-founded demand-side platform (DSP) and ad-serving company — so that it silently rewrote cryptocurrency wallet addresses on every website that loaded it. The compromised file, trackpoint-async.js, is served from Adform's tracking domain s2.adform.net and is embedded, unmodified by end customers, on every site that runs Adform's tag. Independent security researcher Kevin Beaumont publicly disclosed the compromise on his DoublePulsar blog, and a separate researcher, Max Maass, captured and published a full copy of the trojanized script on GitHub Gist the same week [2,3].

The injected code contained two cooperating malicious blocks appended to the end of the legitimate tracker. One monitored clipboard copy events and periodically rescanned the page for text matching Bitcoin, Ethereum, and TRON address formats, replacing any match with an attacker-controlled address; a second block walked the page's DOM and hooked form-field value setters so that addresses typed or pasted directly into input fields, textareas, and contenteditable elements were rewritten as well, with the cursor position restored to hide the substitution [3,1]. Both blocks obfuscated their replacement strings and configuration with a six-byte XOR cipher and beaconed the visited hostname and page path to an external server at 84.32.102[.]230:7744 [3,8].

Adform told reporters it detected the incident and removed the malicious code on 27 July 2026, notified affected clients, and reported the matter to relevant authorities, while cautioning that browsers which had cached the tampered file might continue executing it until the cache was cleared [1]. Beaumont's analysis, based on archived copies of the script, indicates the malicious code had likely been live for roughly a week before removal and evaded detection completely — zero of 61 antivirus engines on VirusTotal flagged the file at the time it was examined [2,7].

Because Adform's tag is embedded across a very large number of downstream sites — the company itself reports ~1,800 direct global customers in its own 2025 annual report, while independent ad-tech technographic tracking firm 6sense measures Adform's script as present on roughly 14,250–14,650 individual domains worldwide, good for ~30–30.5% share of the DSP category by site count and the #2 position in the market — this is a textbook example of the risk concentration created by widely embedded third-party JavaScript: one compromised vendor script becomes a single point of failure for the security of every site, and every visiting browser, that loads it [9,11]. As of this writing, Adform has not published a precise count of affected sites, visitor exposure, or confirmed financial losses, and no independent, on-chain confirmation of stolen funds has been published. This report lays out what is documented, what is inferred, what remains unknown, and what security and ad-operations teams should do in response.

1. What Happened: Narrative and Timeline

Ad-serving and demand-side platform vendors like Adform operate what is, from a security standpoint, one of the largest attack surfaces on the modern web: a JavaScript tag that thousands of unrelated websites load directly into the browsing context of their visitors, generally with the vendor's own domain (here, s2.adform.net) as the trusted origin. When that tag is compromised at the source, every downstream site inherits the compromise without any action, and often without any awareness, on the site owner's part.

The compromised asset is trackpoint-async.js, hosted at https://s2.adform.net/banners/scripts/st/trackpoint-async.js. This is Adform's standard first-party tracking pixel/script, used across customer sites for conversion tracking, cookie and consent management (TCF/GPP), and product/order tracking for e-commerce integrations [3,6].

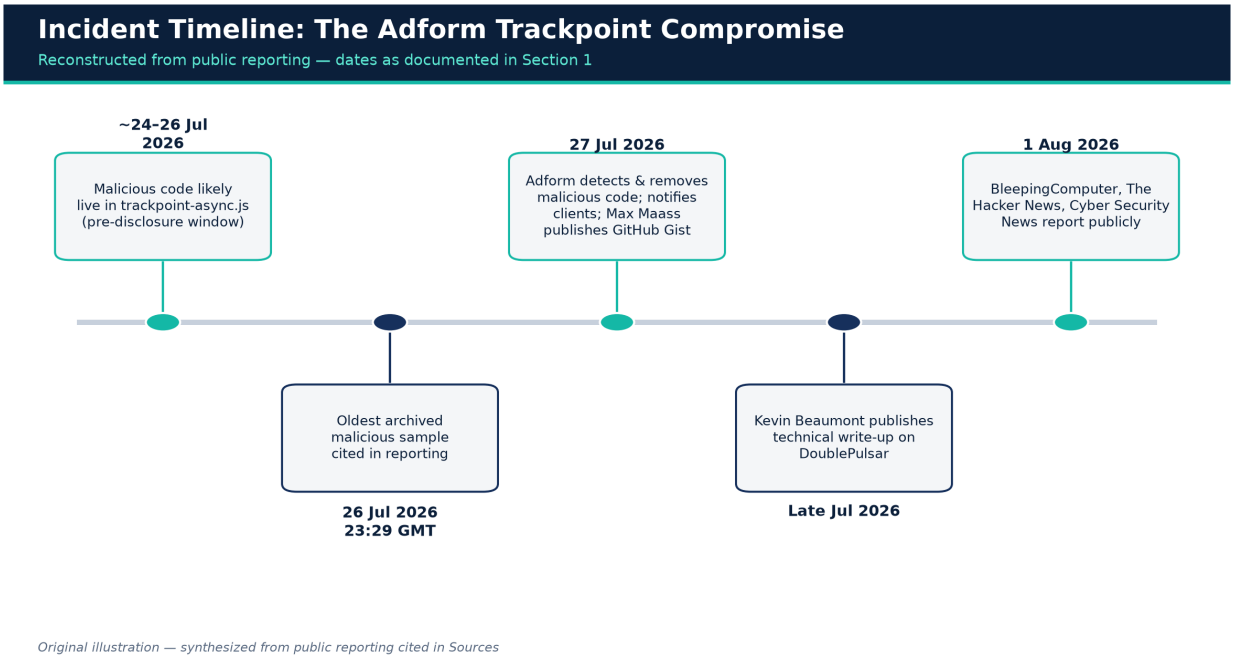

Figure 1. Incident timeline, reconstructed from public reporting.
Original illustration — synthesized from sources [1,2,3,6]

Timeline, as reconstructed from public reporting

- ~24–26 July 2026: Based on archived copies of the script, malicious code appears to have been present in the live file for roughly a week before its public disclosure; the oldest malicious sample cited in reporting was archived on 26 July 2026 at 23:29 GMT [1,2].
- 27 July 2026: Adform states it detected the suspicious activity, removed the malicious code from the script, and began notifying affected clients directly; the company also reported the incident to relevant authorities [1,8]. The same day, researcher Max Maass captured and published a full copy of the trojanized script as a public GitHub Gist, preserving the injected code for community analysis [3,5].
- Late July – 1 August 2026: Kevin Beaumont published a detailed technical write-up on DoublePulsar disclosing the compromise publicly, noting that — to his knowledge at time of writing — Adform had not made a public statement about the incident, even as it had reportedly notified some clients directly [2].

Mainstream security trade press, including BleepingComputer, The Hacker News, and Cyber Security News, picked up the story on 1 August 2026 [1,8,6].

What the malicious code did, at a high level

Any visitor whose browser executed the tampered trackpoint-async.js — which is to say, any visitor to any site embedding Adform's tag while the malicious version was live — had two malicious functions running silently on the page: one watching for a copy of a cryptocurrency address and substituting the attacker's address into the clipboard, and one watching the page and any editable field on it for text that looked like a wallet address and overwriting it in place. If the visitor then pasted or reused that address to send a Bitcoin, Ethereum, or TRON payment, the funds would go to the attacker instead of the intended recipient [8,3].

Figure 2. The compromised Adform script rewriting a cryptocurrency address on a live page.
Credit: BleepingComputer, sourced from Kevin Beaumont / DoublePulsar's analysis [1,16]

Beaumont specifically flagged the persistence of the substitution as the most dangerous property of the attack: because the script continuously rescanned the page and clipboard rather than acting only once, a user who noticed the pasted address looked wrong and tried again would simply have it swapped again. As reported, Beaumont observed: “Even if you notice the address is wrong and recopy the wallet, it keeps replacing it” [6,7].

2. Who Is Adform, and Why Its Compromise Matters at Scale

2.1 Company profile

Adform A/S is a privately held, Copenhagen-headquartered advertising-technology company founded on 17 January 2002, describing itself as an independent, EU-based, full-stack advertising platform spanning demand-side (buy-side), sell-side, ad-serving, and data-management products [9,10]. According to its own 2025 annual report:

- Revenue of EUR 103.4 million in 2025 (roughly flat, +1% year-over-year; ~5% underlying growth excluding non-recurring items in 2024), with an all-time-high EBITDAC of EUR 17.2 million [9].
- ~1,800 global customers, 29 offices in 24 countries, and roughly 700 employees (698 average FTEs in 2025) [9].
- Positioned by its own materials as the #1 EU-based platform and #2 independent DSP globally [9].
- A platform enabling 1.5 billion ad impressions displayed daily, transacting and serving ads in more than 180 countries during 2025, run out of eight global data centers [9].
- In December 2025, Adform acquired Splicky (Goldbach Group's ad-tech division) to expand its DACH-region presence and digital out-of-home capabilities [9,10].

2.2 Reconciling the “1,800 customers” and “~30% DSP market share” figures

Public commentary on this incident cites two scale figures that appear, at first glance, inconsistent: “~1,800 customer sites” and “roughly 30% DSP market share.” Both figures are real, but they measure different things, from different sources, and it is worth being precise about that distinction for anyone assessing blast radius:

- ~1,800 is Adform's own reported customer count — the number of direct contractual relationships (advertisers, agencies, and publishers) the company serves, as disclosed in its 2025 annual report [9,1,8].
- ~14,250–14,650 domains, corresponding to ~30–30.5% share of the DSP technology category and the #2 rank behind StackAdapt (~41%), is a technographic measurement from the market-intelligence firm 6sense, which crawls the live web and counts distinct websites where Adform's DSP-related script tags are detected in page source — not Adform's own count of contracted accounts [11,6,8].

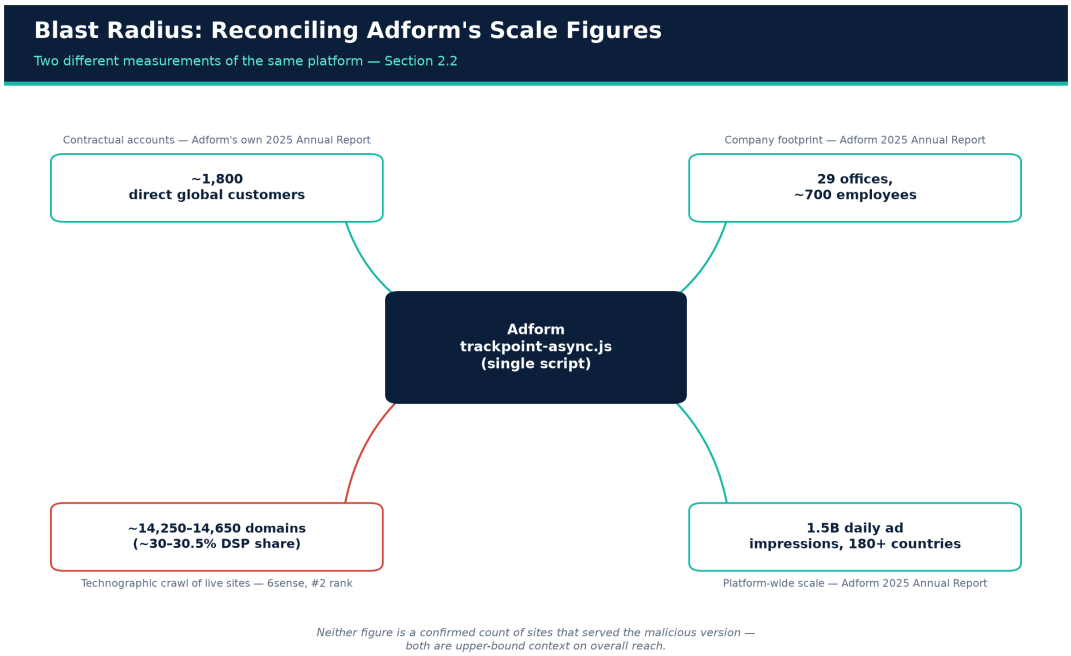

Figure 3. Reconciling Adform's two scale figures: contractual customers vs. technographic domain count.
Original illustration — synthesized from sources [9,11]

The gap between the two numbers (~1,800 vs. ~14,000+) is expected and explainable: a single Adform “customer” (an agency or advertiser account) can run campaigns across many publisher domains, sub-brands, regional storefronts, and landing pages, each of which independently embeds the tracking/serving tag; resellers and managed-service arrangements further multiply the domain count relative to the contract count. For the purposes of incident blast radius, the technographic figure (tens of thousands of domains, tens of millions of daily visiting browsers, given 1.5 billion daily ad impressions company-wide) is the more relevant one — it approximates how many distinct websites' visitors could have loaded the tampered script — while the “1,800 customers” figure is more relevant to Adform's own incident-notification obligations. Neither figure is a confirmed count of sites that actually served the malicious version of the file during the live window, which Adform has not published; both should be read as upper-bound context on the platform's overall reach, not as a precise casualty count for this specific incident.

2.3 Why third-party ad/tracking scripts are a uniquely attractive supply-chain target

Ad-tech and analytics vendors sit in an unusual trust position on the web: their JavaScript typically executes in the same browsing context as the host page (no iframe sandbox), with the same access to the DOM, cookies, and — critically for this incident — clipboard and form-input events, as first-party code. A website owner who embeds s2.adform.net's tag is implicitly trusting Adform's infrastructure and build pipeline as much as their own, and that trust is transitive to every one of that site's visitors.

This is structurally identical to the risk pattern behind other major web supply-chain incidents, most notably the 2024 Polyfill.io compromise, in which a newly acquired CDN domain serving a widely used JavaScript

polyfill library was modified to redirect mobile visitors to scam sites; that single domain take-over affected an estimated 100,000+ websites, prompted Cloudflare and Fastly to stand up trusted mirrors, and led Google to begin disapproving ads on pages still loading the compromised script [12]. The Adform incident differs in payload (cryptocurrency-address substitution rather than malicious redirects) and in root cause (an apparent direct compromise of Adform's own script/serving pipeline rather than a domain resale), but the structural lesson is the same: a single upstream JavaScript dependency, loaded without subresource integrity or sandboxing, can convert one vendor compromise into a mass-casualty client-side attack with no code change required on any individual victim site.

3. Technical Analysis of the Malicious Code

The most detailed technical accounts come from two independent artifacts: a captured copy of the trojanized script published by researcher Max Maass on GitHub Gist [3], and a further mirrored capture circulated by Kevin Beaumont [2] and referenced by Infosec Exchange users as also available via Pastebin [4]. Both captures are consistent with each other on every structural detail below.

3.1 Structure of the file

The malicious file is not a small, purpose-built payload — it is the full, legitimate Adform trackpoint-async.js library (several thousand lines, including cookie management, CryptoJS and JSEncrypt cryptographic routines used for legitimate consent-string signing, TCF/GPP consent-framework handling, and e-commerce order/product tracking) with two additional malicious blocks appended to the end of the file [3]. This “append, don't replace” approach is a classic supply-chain-implant technique: the legitimate script continues to function exactly as before (so the compromise produces no visible breakage that would prompt investigation), while the appended code executes silently alongside it.

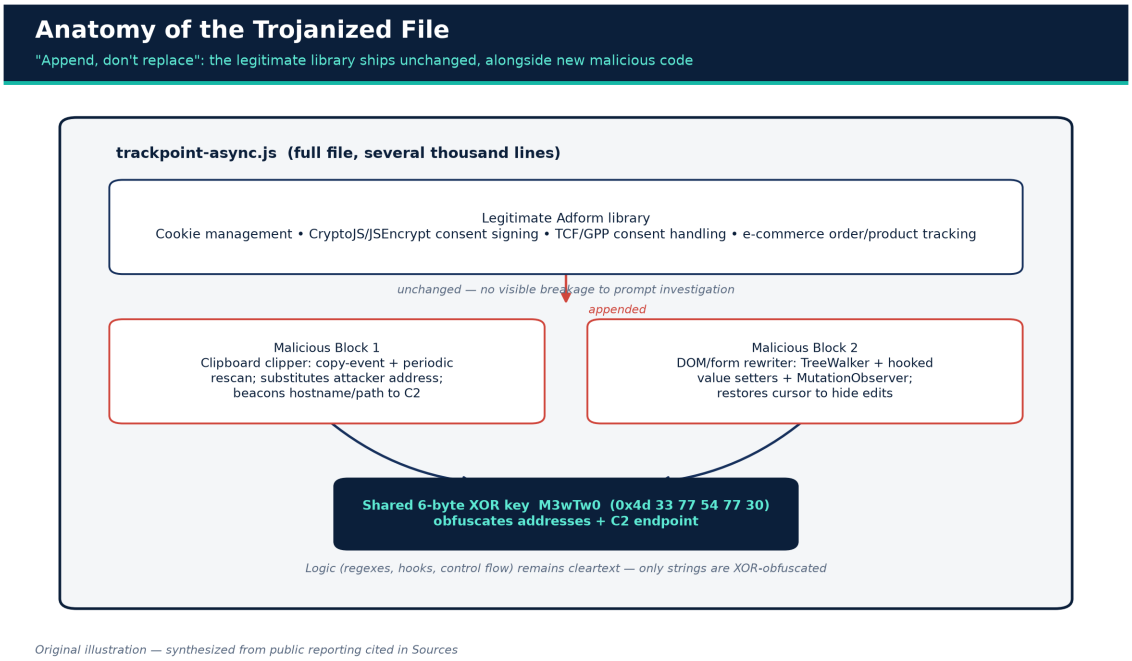

Figure 5. Anatomy of the trojanized trackpoint-async.js file.
Original illustration — synthesized from source [3]

3.2 Obfuscation

Both malicious blocks obfuscate their sensitive strings — principally the attacker's replacement wallet addresses and the exfiltration endpoint — using a six-byte XOR key, reported identically across both independent analyses as the byte sequence [0x4d, 0x33, 0x77, 0x54, 0x77, 0x30] (ASCII M3wTw0) [3]. A small decode routine in the script XORs each character of the obfuscated string against this repeating key at runtime to reveal the plaintext replacement address or beacon URL. This is a lightweight obfuscation scheme — it defeats naive string-based static signatures but is trivially reversible by any analyst who extracts the key, and does not obfuscate the logic of the script (the regexes, event hooks, and control flow remain in cleartext), which is likely why analysts were able to fully reconstruct its behavior within days.

3.3 Wallet-address detection (regular expressions)

Both malicious blocks use the same three regular expressions to identify cryptocurrency addresses appearing in clipboard content, page text, or form fields [3]:

Currency	Pattern	Matches
Bitcoin	`\b(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,62}\b`	Legacy P2PKH (1...), P2SH (3...), and native SegWit bech32 (bc1...) address formats
Ethereum	`\b0x[a-fA-F0-9]{40}\b`	Standard 20-byte hex EVM addresses (also matches BNB Smart Chain, Polygon, and other EVM-compatible chains using the same format)

Currency	Pattern	Matches
TRON	`\bT[1-9A-HJ-NP-Za-km-z]{33}\b`	Base58Check TRON addresses beginning with T

These are broad, format-level pattern matches (they do not validate checksums), meaning the script would flag — and could rewrite — any string on the page or in the clipboard that merely looks like a valid address in one of these three formats, regardless of whether it was actually a real, currently-used destination address.

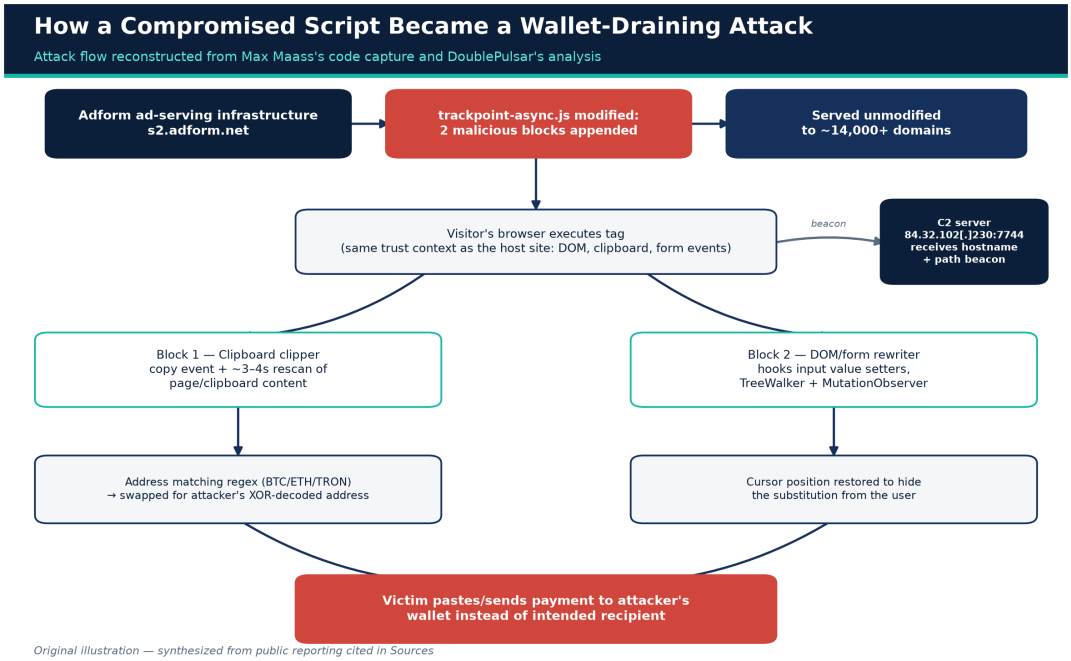

Figure 4. Attack flow: from compromised script to victim wallet drain.
Original illustration — synthesized from sources [3,8]

3.4 Malicious Block 1: clipboard-focused clipper

The first appended block is a comparatively compact clipboard-hijacking routine. Per the code analysis, it:

- Attaches a listener to the browser's copy event and to a periodic timer (reported as roughly every 3–4 seconds across the two source analyses) that inspects available clipboard/page content for text matching the regexes above [3,8].
- On a match, substitutes the attacker's XOR-decoded wallet address for the corresponding cryptocurrency, so that a copy action captures the attacker's address instead of (or immediately after) the user's intended one.
- On page load, sends an HTTP request to the hardcoded command-and-control endpoint at 84.32.102[.]230:7744, of the form http://84.32.102[.]230:7744/p?h=<hostname>&u=<path> — reporting the visited site's hostname and URL path back to the attacker, functioning as a lightweight victim-tracking/reconnaissance beacon [3,8].

A note on browser clipboard-permission behavior. Modern browsers restrict arbitrary, unprompted programmatic reads of the system clipboard via the asynchronous navigator.clipboard.readText() API — such calls normally require a user gesture and, in many contexts, an explicit permission grant, and will otherwise throw or silently fail. Multiple secondary summaries describe this block as “polling the clipboard every few seconds,” which is the correct external behavior but is likely implemented via a copy-event listener combined with periodic page-content rescanning, rather than true out-of-band OS clipboard polling. We flag this as a point where the primary code captures are more authoritative than tertiary reporting, and where a reader auditing the raw script directly should verify the exact API calls rather than relying on any single paraphrase, including this one.

3.5 Malicious Block 2: DOM- and form-field-level rewriter

The second, more extensive block goes beyond the clipboard to attack any wallet address that appears anywhere on the page or is typed directly into a form, which defeats the natural user instinct to “just retype the address instead of pasting it.” Per the code analysis, this block:

- Uses a TreeWalker to iterate all text nodes in the document, scanning for and rewriting matches to the three address regexes wherever they appear in rendered page content [3].
- Hooks the native property setters on HTMLInputElement.prototype.value and HTMLTextAreaElement.prototype.value, so that even programmatic writes to a form field (not just user typing) pass through the rewriting logic.
- Listens for copy, cut, paste, and input events on the page, rewriting matched addresses at each of these interception points.
- Monitors contenteditable elements directly, and uses a MutationObserver configured for childList, subtree, and characterData changes on document.body to catch dynamically injected or edited content (e.g., single-page-application re-renders) that the initial tree walk would have missed.
- Restores cursor/caret position after a text substitution, specifically to avoid the visible “jump” that would otherwise tip off an attentive user that their input had just been altered [3].
- Re-scans on a fixed interval (reported as every 3 seconds) in addition to event-driven triggers, which produces the persistence behavior Beaumont highlighted — a corrected or re-pasted address is simply overwritten again on the next scan cycle.

Both blocks target the same three currencies and share the same XOR key, suggesting either a single author writing two complementary components, or a rapid iteration where a first, simpler clipboard-only clipper was extended shortly after with a more thorough DOM/form-field variant — the public analyses do not resolve which.

3.6 Detection evasion and antivirus results

At the time researchers examined the captured sample, it registered zero detections across 61 engines on VirusTotal [7,1]. This is unsurprising for browser-JavaScript-based clipboard/DOM attacks: most antivirus

and endpoint-detection engines are tuned to detect binary malware and known malicious URLs/hashes, not to statically analyze arbitrary first-party-hosted JavaScript for clipboard-manipulation logic, and the XOR-obfuscated strings would in any case defeat simple literal-string matching against known bad wallet addresses or IPs.

3.7 Attacker infrastructure and attribution

Public reporting identifies a single command-and-control/exfiltration endpoint, 84.32.102[.]230:7744, used by both malicious blocks to receive victim hostname/path beacons [3,8]. No public reporting as of this writing attributes the intrusion to a named threat actor or group, discloses how Adform's build/delivery pipeline for trackpoint-async.js was initially compromised (e.g., stolen CI/CD credentials, a compromised internal developer account, a compromised CDN/edge configuration, or a compromised third-party dependency inside Adform's own build), or confirms whether any funds were actually diverted to attacker-controlled wallets. These are the most significant open questions in the public record (see Section 7, Limitations).

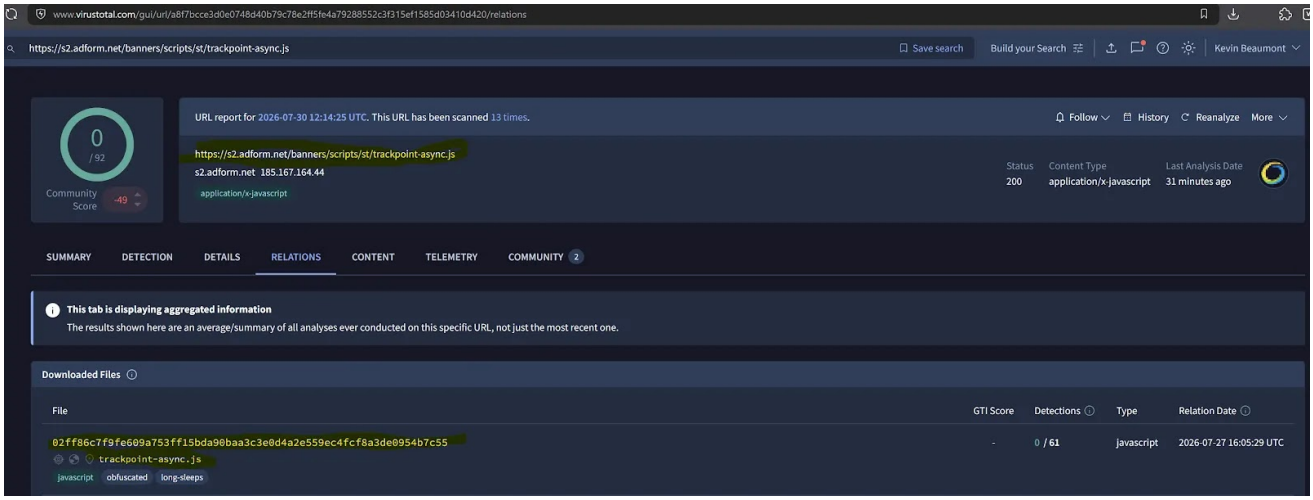

Figure 6. The malicious payload block appended to the end of the legitimate tracking script.
Credit: Cyber Security News, sourced from Kevin Beaumont's research [6,17]

4. Indicators of Compromise

The following IOCs are drawn from the two independent code-capture analyses and corroborating trade-press reporting. Defenders should treat this as a starting point for hunting, not an exhaustive list — Adform has not published its own IOC bulletin as of this writing.

Type	Value	Source
Compromised script URL	`https://s2.adform.net/banners/scripts/st/trackpoint-async.js`	[6,8]
Malicious file SHA-256	`02ff86c7f9fe609a753ff15bda90baa3c3e0d4a2e559ec4fcf8a3de0954b7c55`	[6,7]

Type	Value	Source
Exfiltration/C2 IP + port	`84.32.102[.]230:7744`	[3,8]
Beacon URL pattern	`http://84.32.102[.]230:7744/p?h=<hostname>&u=<path>`	[3]
XOR obfuscation key	`4D 33 77 54 77 30 (6 bytes)`	[3]
Bitcoin address regex	`\b(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,62}\b`	[3]
Ethereum address regex	`\b0x[a-fA-F0-9]{40}\b`	[3]
TRON address regex	`\bT[1-9A-HJ-NP-Za-km-z]{33}\b`	[3]
Earliest known malicious sample	`26 July 2026, 23:29 GMT`	[1]
VirusTotal detection at analysis time	`0 / 61 engines`	[7]

Recommended defensive action on these IOCs: block outbound connections to 84.32.102.230:7744 at the network egress layer; hash-check any cached or CDN-mirrored copies of trackpoint-async.js against the known-bad SHA-256 above (and, more robustly, diff against Adform's current published version); and treat any site that was serving this file between approximately 24 July and 27 July 2026 as having potentially delivered the payload to its visitors during that window.

5. Adform's Response, and What Customers and End Users Should Do

According to statements provided to reporters, Adform said it detected the malicious modification and removed it on 27 July 2026, notified affected clients, and reported the incident to the relevant authorities [1]. The company characterized the malware as not designed to install persistent software on a visitor's device, stating it “was not designed to install software on a user's device or establish persistence” and operated only for the duration that an affected page remained open in the browser [1,6]. On the question of whether visitor data (beyond wallet addresses) had been exfiltrated, Adform's notice reportedly stated that “technical analysis indicates that such transmission may have been possible,” while noting the company had found no evidence that it had actually occurred [6] — a carefully hedged statement that leaves open the possibility of broader data exposure beyond the address-swap payload described above.

Adform's stated recommendations to affected parties, as relayed in press coverage, were:

- Clear browser cache, because the tampered script may remain cached locally and continue executing even after Adform removed the malicious version from its servers [1,8].
- Independently verify any cryptocurrency wallet address — character-by-character, or via a second out-of-band channel — before sending funds, rather than trusting a pasted or displayed address at face value [8].

As of this writing, no public reporting indicates Adform has published: a full IOC bulletin, a root-cause statement explaining how the script's build or delivery pipeline was initially compromised, a count of affected sites or visitors, or confirmation/denial of actual financial loss to end users. Beaumont's original disclosure noted that, to his knowledge at the time, Adform had not made any public statement about the incident even as direct client notifications were reportedly underway [2] — meaning the primary source of public technical detail on this incident has been independent researchers, not the vendor.

Practical guidance: for website operators and ad-operations teams

- Treat this incident as confirmation that any third-party script loaded without Subresource Integrity (SRI) or sandboxing executes with the full trust and privileges of your site, including access to clipboard events, form inputs, and the DOM of pages that may process sensitive user actions (payments, wallet addresses, credentials). Classic SRI is difficult to apply to frequently-updated, vendor-controlled tags like ad-serving scripts, but options worth evaluating include: script-monitoring/tag-management tools that alert on unexpected changes to the byte content of third-party scripts; loading vendor scripts inside a sandboxed iframe with a restrictive Permissions-Policy where the vendor's functionality allows it; and Content-Security-Policy script-src allowlisting scoped as narrowly as possible.
- If your site displays or processes cryptocurrency payment addresses (e-commerce checkouts, donation pages, exchange interfaces) and also loads third-party ad or analytics tags, treat that combination as elevated risk and consider isolating the payment-address-rendering surface from pages that load broad third-party tag stacks.
- Confirm with your ad-tech/DSP vendors (Adform or otherwise) what integrity-verification and change-notification mechanisms they offer for tags you embed, and ask specifically whether they support SRI-compatible versioned script URLs rather than a single mutable “latest” endpoint — mutable, unversioned vendor script URLs are precisely what makes this class of attack possible.

Practical guidance: for end users who may have visited an affected site between ~24–27 July 2026

- Clear browser cache per Adform's guidance, since a locally cached copy of the tampered file could continue to execute after server-side remediation [1].
- Manually verify, character-by-character, any cryptocurrency address before sending funds — ideally by comparing it against an address obtained through a second, independent channel (a QR code, a previously-saved contact in your wallet's address book, or a phone confirmation), rather than trusting a freshly pasted or on-page-displayed string.
- Where supported, use hardware wallets or wallet software with on-device address display and confirmation, since the malicious code operates in the browser DOM/clipboard layer and cannot alter an address shown on an air-gapped hardware device's own screen.
- If you completed a cryptocurrency transfer sourced from or through an affected site during the exposure window and the destination does not match your intended counterparty, treat it as a suspected theft:

preserve transaction hashes and timestamps, and consider reporting to the relevant national cybercrime-reporting body (e.g., the FBI's Internet Crime Complaint Center, IC3, in the United States) alongside any exchange or platform involved in the transfer.

6. Broader Context: Clipboard Hijacking as an Established, Growing Crime Pattern

The Adform incident is a browser-based, third-party-script-delivered variant of a well-established malware category generally called “clipper” malware — software that monitors the system or application clipboard and substitutes a cryptocurrency address the moment a victim copies one. Historically, clippers have most often been distributed as standalone desktop or mobile binaries rather than through a trusted advertising vendor's JavaScript, which makes the Adform case notable for its distribution mechanism even though the underlying substitution technique is not new.

For scale and comparison, Check Point Research published a detailed analysis (17 June 2026) of an active Rust-based clipper distributed via fake “profit tools” (Solana/Pump.fun sniper bots, crash-game predictors, cracked wallet utilities) promoted through a coordinated fake-reputation campaign spanning GitHub, SourceForge, and a dedicated YouTube channel using AI-generated narrators and inflated engagement metrics [13]. That campaign's Windows binary alone embedded more than 15,500 attacker-controlled wallet addresses (roughly 15,000 Bitcoin addresses across bech32/legacy/P2SH formats plus around 500 Ethereum addresses) so the malware could present a plausible-looking substitute for whatever currency and address format it detected, and supported detection of 18+ cryptocurrency address formats; a macOS variant used a shell-script Gatekeeper bypass and a LaunchAgent for persistence [13]. The Adform malware is architecturally simpler by comparison — it uses one or a small number of hardcoded replacement addresses per currency rather than a rotating pool of thousands, and, being purely in-browser JavaScript, has no OS-level persistence and stops functioning as soon as the affected tab is closed, consistent with Adform's own characterization of the malware's limited persistence [1]. What the two campaigns share is the underlying economic logic — cryptocurrency transfers are irreversible once confirmed on-chain, making address substitution an unusually low-friction, high-yield theft technique compared to attacks that require credential theft, account takeover, or defeating multi-factor authentication.

At the macro level, the U.S. Federal Bureau of Investigation's Internet Crime Complaint Center (IC3) reported $11.366 billion in cryptocurrency-related fraud losses across 181,565 complaints in 2025, a roughly 22% increase over the prior year, with cryptocurrency investment fraud alone accounting for $7.228 billion of that total across 61,559 complaints [14,15]. IC3's public cryptocurrency-crime guidance page also specifically warns the public about the general risk pattern of manipulated wallet addresses and urges independent verification before transacting [15], though IC3's public materials as reviewed for this report do not break out a separate loss figure specific to clipper/address-substitution malware as distinct from investment-scam and romance-scam categories — a genuine gap in the available statistics that limits precise sizing of this specific threat vector's aggregate financial impact.

The Polyfill.io precedent (Section 2.3) remains the closest prior analogue in the ad-tech/web-supply-chain space for scale of exposure — over 100,000 affected sites from a single compromised CDN-hosted script, prompting emergency mirror infrastructure from Cloudflare and Fastly and enforcement action from Google Ads [12] — though its payload (malicious redirects to scam and gambling sites) differed from Adform's targeted financial-address substitution.

7. Limitations of the Public Evidence

This report is built entirely from public reporting, independent researcher analysis, and Adform's own corporate disclosures (its published annual report); at the time of writing, Adform had not published a dedicated incident report, security advisory, or IOC bulletin of its own, and no law-enforcement or regulatory finding on the incident is yet public. Readers should weigh the following gaps explicitly:

- No confirmed victim or loss count. No source reviewed for this report provides a number of end users affected, a number of confirmed successful address substitutions, or a dollar/cryptocurrency amount actually stolen. All scale figures in this report (customer counts, market share, daily impressions) describe Adform's overall platform reach, not the confirmed footprint of this specific incident.
- No confirmed root cause. Public reporting establishes what the malicious code did and where it was served from, but not how attackers gained the ability to modify trackpoint-async.js in the first place.
- No attacker attribution. No source reviewed attributes the intrusion to a specific individual, group, or nation-state; the only infrastructure identified is the single C2 IP/port pair used for beaconing.
- Discrepant secondary reporting on discovery date. While most sources converge on Adform detecting and removing the code on 27 July 2026, at least one secondary source dates “discovery” to 30 July 2026 [7]; this report treats the 27 July date as most likely correct because it is corroborated by multiple independent outlets citing Adform directly, but flags the discrepancy rather than silently resolving it.
- Technical mechanism details drawn from secondary paraphrase. Some specific implementation claims (e.g., “reads the clipboard every 3–4 seconds”) are paraphrased across several tertiary security-news write-ups rather than quoted verbatim from the primary code captures; Section 3.4 flags one place (asynchronous Clipboard API permission behavior) where the precise technical mechanism deserves direct verification against the raw script.
- No independent on-chain forensic confirmation. No source reviewed traces the attacker's 84.32.102.230-associated wallet addresses on a block explorer to confirm receipt of stolen funds; this report does not claim any specific dollar amount was stolen, because no such figure is publicly documented.

Given these gaps, readers — particularly incident responders and journalists — should treat this as an evolving story: Adform's eventual public post-incident report, if published, and any subsequent law-enforcement action, should supersede the reconstruction presented here wherever they conflict.

Sources

Every citation in this report links to one of the numbered sources below. Click a number in the text, or the source name here, to open the original material.

1	BleepingComputer — "Online ad firm Adform's script compromised to steal cryptocurrency"
2	Kevin Beaumont, DoublePulsar — technical disclosure
3	Max Maass, GitHub Gist — captured malicious script
4	Pastebin capture referenced in researcher discussion
5	Max Maass, Infosec Exchange — researcher disclosure
6	Cyber Security News — "Hackers Turned a Trusted Advertising Platform Into a Crypto-Stealer Delivery Network"
7	IT-Connect — "Adform breach: ad script steals cryptocurrency for at least a week"
8	The Hacker News — "Hackers poison Adform script to swap crypto wallets"
9	Adform A/S, Annual Report 2025 (PDF)
10	Wikipedia: Adform
11	6sense — Adform DSP Market Share (technographics)
12	BleepingComputer — Polyfill.io supply-chain compromise
13	Check Point Research — "From Stars to Upvotes: Fake Reputation Fueling a Crypto Clipboard Hijacker"
14	FBI Internet Crime Complaint Center (IC3), 2025 Annual Report (PDF)
15	FBI IC3 — Cryptocurrency crime information page
16	Visual credit: BleepingComputer (sourced from Kevin Beaumont / DoublePulsar)
17	Visual credit: Cyber Security News (sourced from Kevin Beaumont's research)