The Hidden Relay: Inside a Device Code Phishing Kit That Never Shows the Code

KnowBe4 Threat Lab | Nov 12, 2026

Post

Lead Analysts: Maddhav Grandy, Prabhakaran R and Jeewan Singh Jalal

A smart TV or a streaming box throws a short code on screen and tells us to go type it into a website on your phone to finish signing in. It feels almost quaint compared to typing a password.

Microsoft built that same convenience into its own sign-in flow, for devices that cannot easily take a password, handing the user a short `user_code` to approve on a separate screen. Device code phishing takes that convenience and turns it into bait: phishing kits impersonate the device, get handed the code, and trick the victim into approving it on Microsoft's real site instead. Once the victim approves the sign-in and satisfies MFA, Microsoft issues access and refresh tokens straight to the attacker. From there, the attacker can access whatever resources those tokens grant in Microsoft 365, such as email and cloud files.

Device code phishing has risen since August 2024. Microsoft had tracked Storm-2372 using the Microsoft Authentication Broker client ID in device code sign-ins, disclosing the

Subscribe the KnowBe4 Blog

Email

Enter Your Emai

Notification Frequency

- Instant
- Daily
- Weekly
- Monthly

Subscribe



Posts By Topic

Account Takeover

Agent Security

AI

campaign in mid-February 2025. The technique abuses Microsoft's authentic flow to steal access tokens. Victims enter real codes on genuine Microsoft sites, authorizing attacker sessions. The EvilTokens phishing-as-a-service (PhaaS) kit has been observed to commercialize this approach.

Every documented variant of the device code phishing has depended on one step: the attacker opens an OAuth 2.0 device authorization grant, their phishing page hands the victim a short code and the victim copies it into a real Microsoft sign-in page.

The phishing URL we observed in August 2026 never sent that code. The victim types their email address, and the next screen is a genuine Microsoft login page. They just log into their account normally. No code, no instruction to go somewhere and enter. But still the attacker's backend was already holding everything it needed to collect the tokens.

Key Takeaways

- **Zero Code Display:** The attack observed was a device code phishing flow. A user code came back in the kit's response, but it was never shown to the victim, and the flow carried on without it, leaving the attacker positioned to collect tokens if the victim finished signing in.

- **Direct Authentication with Microsoft:** The victim types an email address on the kit's relay page, and the browser lands on login[.]microsoftonline[.]com, ending at Microsoft's own device-approval page.

- **Pre-Staged State vs. AiTM:** Unlike Adversary-in-the-Middle (AiTM) reverse proxies, which relay traffic live as the victim interacts with the page, this kit opens a Microsoft sign-in session in advance and embeds that session's tokens into the relay page's HTML before the victim submits anything. The session is already open and waiting, the victim's action only carries it forward. The pre-staged private session

AI Risk Management
AIDA
Antivirus
Bank and Finance
Brand Impersonation
Business Email Compromise
CEO Fraud
View all topics >

tokens are:

1. *ctx* - a token that identifies which sign-in session you're currently in.

2. *flowToken* - a token that changes at every step, so Microsoft can tell the sign-in steps are happening in the right order.

3. *canary* - a token that ties a submitted form to the exact page that issued it, so a mismatched submission gets rejected.

- **Passkeys and FIDO2 Do Not Stop This:** Authentication happens entirely on Microsoft's genuine domain, so FIDO2 and passkey origin checks pass without error. The session state even carries an essential 'ngcmfa' claim; which Microsoft defines as "equivalent to MFA (multi-factor authentication), used for provisioning of certain advanced credential types." A new login session is requested even if there is an existing session. The sign-in must use Microsoft Authenticator push or Windows Hello for Business, and the session is restricted to a strict 10-minute validity window. That leaves the attacker's already open authorization grant positioned to receive the tokens.

How Does the Attack Work?

KnowBe4 ThreatLabs observed a generic phishing email urgently requesting approval of a non-disclosure agreement (NDA). The analysis of the phishing URL led to discovery of this attack.

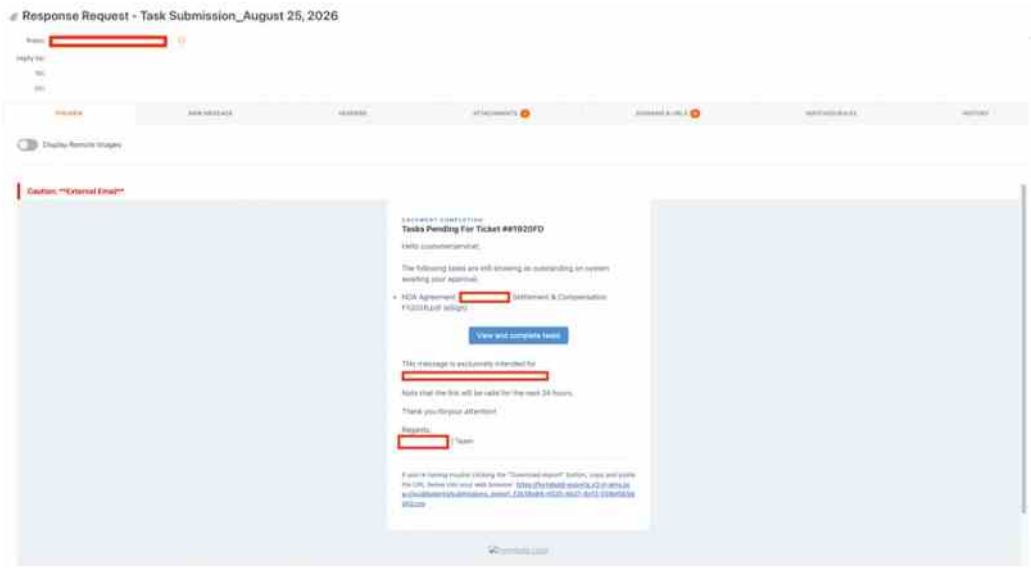


Figure 1: NDA themed lure

The attackers used legitimate infrastructure to send these phishing emails, in this case Mandrill, Mailchimp's transactional email service. The message arrived fully authenticated from a sender with a clean reputation, which helped in evading Secure Email Gateways (SEGs).

Anatomy of the Attack Chain

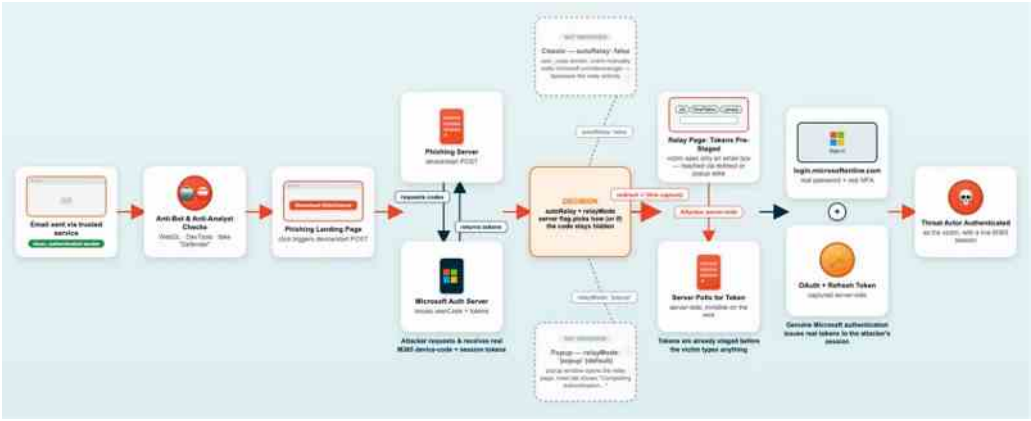


Figure 2: Device Code Hidden Relay Kill chain

The phishing URL was wrapped in Mandrill's click tracking service, which hides the real destination from email scanners. The URL then redirected through several hops before reaching the final phishing page. Most hops were hosted on serverless infrastructure, such as Cloudflare Workers page.

Microsoft's sign-in endpoint communicates through TCP protocol, and here the Cloudflare Workers page leads the sign-in to happen over QUIC protocol: "*a modern transport*

layer network protocol that speeds up web connections and reduces latency. It serves as the foundation for HTTP/3". It operates on UDP port 443 with transport-layer encryption. Since most inspection tools target TCP for HTTP/1.1 and HTTP/2, they see little of an HTTP/3 session, which Cloudflare serves by default.

The final landing page runs two checks in its own JavaScript before it will start the device code flow, and both are looking for an analyst:



Figure 3: Anti-analysis detection kit

- A WebGL software-renderer check, which sends anything that looks like a sandbox to `microsoft.com`.
- A sweep for automation globals and headless User-Agent strings, plus a DevTools timing trap. Any hit wipes the page.

Only after those passes does the victim see anything, and what they get is a fake "Microsoft Defender" verification overlay. Clicking verify runs a 950 millisecond spinner, then the overlay fades out. Nothing is verified. It is there to make the page feel like it came from Microsoft. Then the page offers a "Download Attachment" button, and clicking it starts the device code flow.

How Does the Device Code Flow Start?

The flow starts with a POST request sent to the Cloudflare Worker hosted phishing URL and the response has the details on how to proceed with the flow. The JavaScript code uses the response to follow the flow intended by the attacker.

```
// POST request related code made from the browser to start the device-code flow no user action required

const response = await fetch(`${API_ENDPOINT}/device/start`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json', 'X-Worker-Token': WORKER_TOKEN },
  body: JSON.stringify({ deviceName: 'Document.pdf', lureType: 'outlookv2' })
});
```

Figure 4: code snippet to start the device code flow

The values sent in POST request are hardcoded, lureType selects the next page template to load, in this case server was instructed to load a fake Outlook-themed login page.The term deviceName is what the victim sees on the next page as the document being requested, in this case Document.pdf.

The response is as shown below (obtained by decrypting the HTTP/3 capture with the sandbox's own TLS session keys).


```
{
  "success": true,
  "autoRelay": true,
  "relayMode": "redirect",
  "sessionId": "ses_1787743590242_9f70b882",
  "relayToken": "372b56d018b8cbc359b0819e402f11d5",
  "userCode": "EGEGABPRP",
  "verificationUriComplete": null,
  "expiresIn": "900"
}
```


Figure 5: Decrypted HTTP/3 capture

> **`"userCode":`** It shows a nine-character uppercase code **`"EGEGABPRP"`**, with a 900 second lifetime, which is 15 minutes. Both the format and the lifetime line up with RFC 8628, the specification Microsoft's authorization server follows.
>
> **`"verificationUriComplete":`** This field is set as "`null,`" which determines that the user has to manually enter the code into the `microsoft[.]com/devicelogin` page rather than endpoint sending a URL where code is already embedded, so the user only has to authenticate.
>
> **`"autoRelay"`** and **`"relayMode":`** These are set as "true" and "redirect," which determines the next steps that should be taken by Cloudflare Worker hosted JavaScript code.

The JSON response mentioned above is being used by same JavaScript code that has sent the POST request:

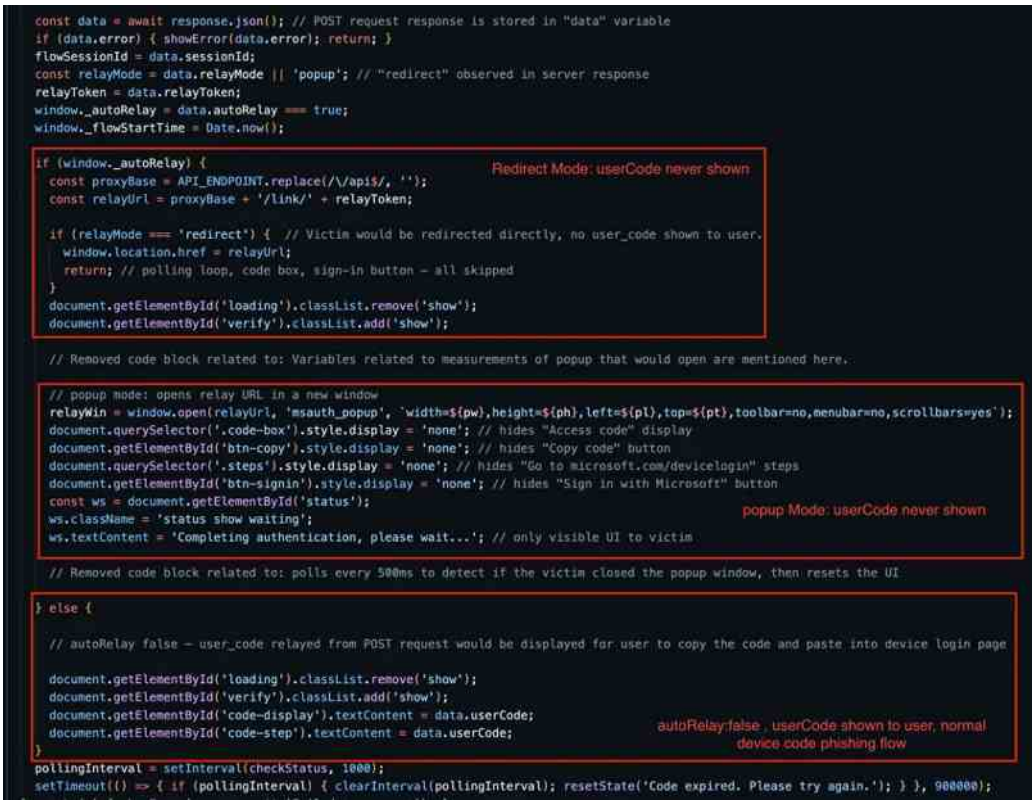


Figure 6: code snippet of autoRelay module

AutoRelay and RelayMode, acts as the flags that decides which of these behaviors the JavaScript runs next.

The kit has three ways to run the device code flow:

1. **Device code Hidden Relay (through redirect)**
In this case, autoRelay is set to 'true' and relayMode as '*redirect',* so instead of showing a code, the JavaScript builds a URL from the relayToken in the response and redirects the victim's browser there.

2. **Device code Hidden Relay (through popup)**
If autoRelay is set to '*false*', the JavaScript writes the userCode into the page, tells the victim where to go and enter it, and starts a loop that polls the backend once a second to ask whether the grant has been approved yet.

3. **Normal Device Code**
If relayMode were missing from the response, the JavaScript defaults to create a popup window and proceed with the Hidden Relay flow.

We observed the first of these, the redirect version. A *userCode* did come back in the response with a 900 second lifetime, so the code existed behind the scenes and it was just never written into the page for the victim to see, and no popup ever opened.

What Does the Victim See Instead of Code?

The browser was sent to the relay page, built from the `relayToken` in the response, at `bandgdruback1[.]workers[.]dev/link/372b56d018b8cbc359b0819e402f11d5`.

The page is titled "Verify your identity" and carries Microsoft branding. It shows the filename `Document.pdf`, the same string the kit hardcoded into the POST body, and it requests for a user email address for validation.

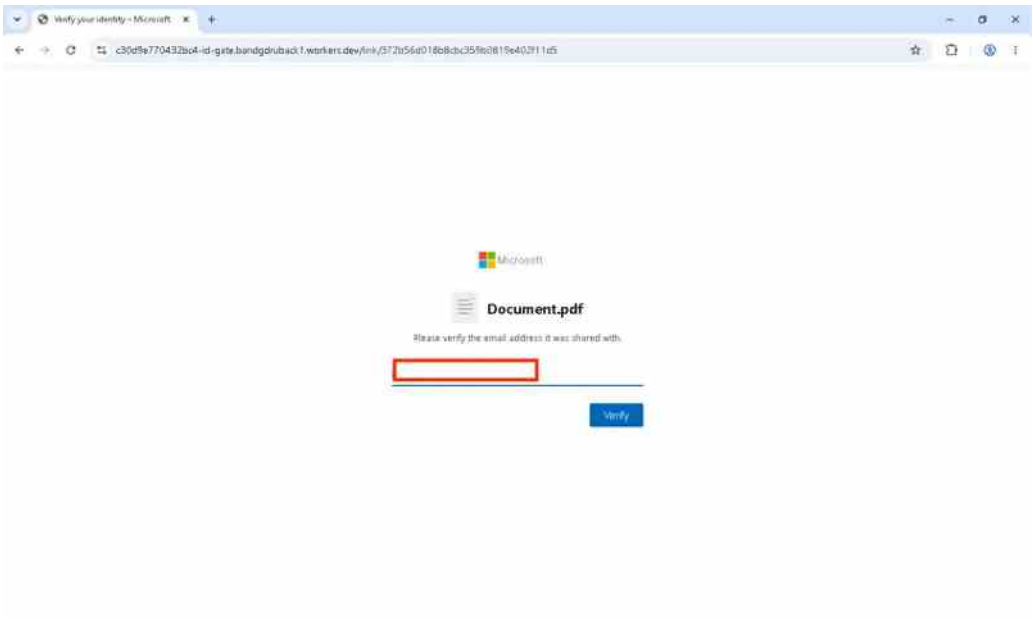


Figure 7: Microsoft themed Final landing page harvesting victim email address

Once the user submits the email address, the browser is handed off to Microsoft.

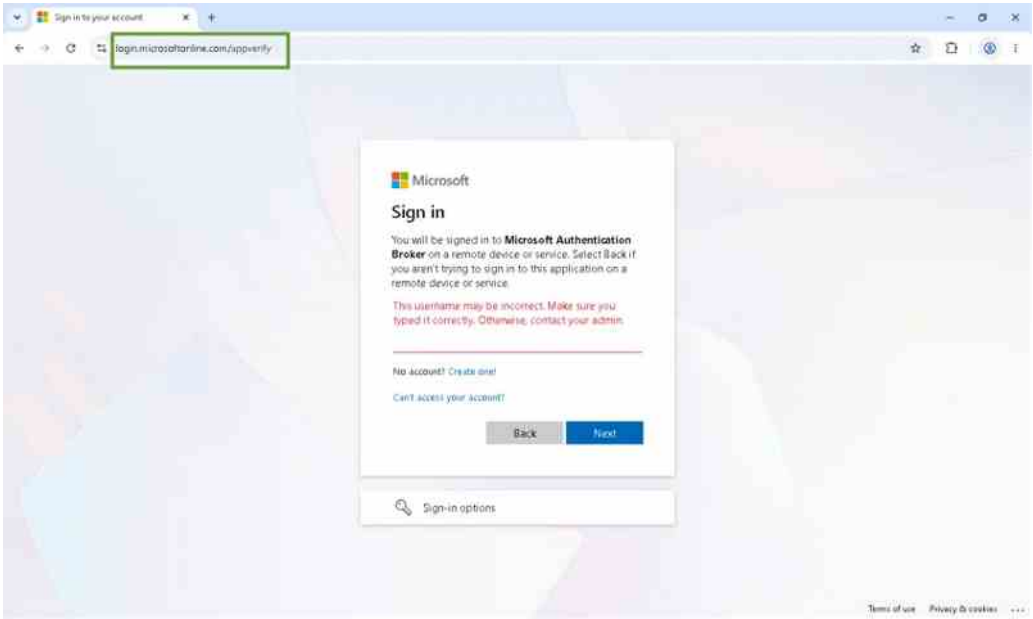


Figure 8: Microsoft Authentication Broker authenticating a Remote Device login

What comes back is login[.]microsoftonline[.]com. The domain and sign-in page belong to Microsoft, and that is where the password prompt and the MFA prompt appear. Nothing the attacker controls is between the victim and their identity provider at this point.

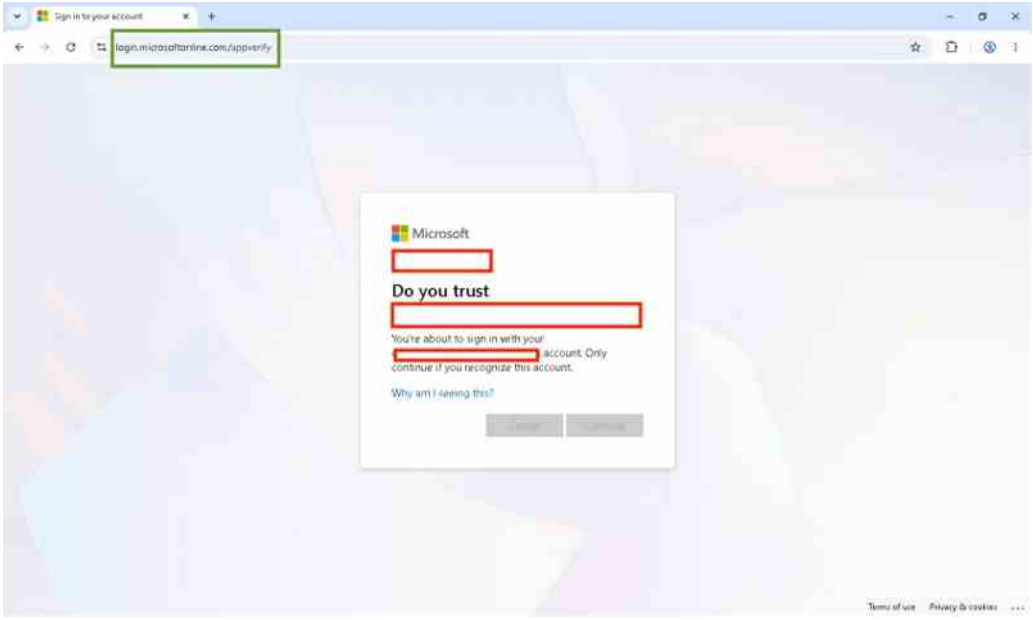


Figure 9: Microsoft Authentication Broker confirming the login

The application mentioned on the consent screen is Microsoft Authentication Broker, and the attacker is targeting this client for the grant.

Where Did Microsoft's Session Parameters Come From?

The webpage shows one email input box, and the source code reveals four more fields that the victim never sees.

```
<!-- The form POSTs to the attacker's Worker. ctx, flowToken, and canary are hidden fields
     pre-staged by the backend — the victim never sees them, but they travel with the submit. -->
<form method="POST" action="/link/372b56d018b8cbc359b0819e402f11d5/submit">

  <!-- ESTS session context — pre-staged, encodes the full auth session state -->
  <input type="hidden" name="ctx" value="rQQIARAAFcrNy9t0AMDxds_2OB7nfPacBC9SPYiue5rmrSkMzVuf5q1p0iZpKxJ-zXuaNG

  <!-- ESTS flow token — per-session replay protection, rotates at each auth step -->
  <input type="hidden" name="flowToken" value="BgABIQEAAAAdDD7nC9b5Q7JPd_okEQRFRXZvU3RzQXJ0aWZhY3RzAQAAAAAArh6j6

  <!-- CSRF canary tied to this ESTS session -->
  <input type="hidden" name="canary" value="9RAjzouXBPdZz5aU7lL1ZQcDZfWxawdSRfgO5kAvNjA=4:1:CANARY:/4E5gFW1Qb+g0

  <input type="hidden" name="type" value="11">

  <!-- The only visible input — victim types their email here, unaware the session is already open -->
  <div class="field-wrap">
    <input type="email" id="loginEmail" name="login" required autofocus autocomplete="off" placeholder="Email ad
  </div>
  <div class="row"><button type="submit">Verify</button></div>
</form>
```

Figure 10: Hidden Microsoft Tokens in the Attacker’s landing page

'`ctx`', '`flowToken`' and '`canary`' are Microsoft's own sign-in session parameters. Microsoft's authentication pages emit them, and no attacker generates them.

These fields are also observed in a reverse proxy phishing kit because it sits in the middle of a live session passing elements from the Microsoft page. But in this case we observe that the tokens arrived embedded in the HTML before the victim interacted with the phishing webpage. That is what rules out a reverse proxy here. A proxy has nothing to put in the page until the victim acts, because its tokens come from a live session it is relaying.

The `'ctx'` token then tells us which session these belong to. It is not encrypted, only encoded, base64url with a six byte header and then a raw DEFLATE stream, so it inflates back to plaintext. Inside it sits an 810 byte Microsoft flow token.

When the `'ctx'` from the attacker-served page and the `$Config.s'ctx'` from the genuine `login[.]microsoftonline[.]com` page reached after submitting the email were both decoded, the same 810 byte token was present in each, byte for byte.

Where the token was found	Length	SHA-1 (first 12 hex)
Inside decoded `'ctx'`, attacker relay page	810	`9abe6d45ae29`
Inside decoded `$Config.s'ctx'`, Microsoft sign-in page	810	`9abe6d45ae29`

So the attacker's page and Microsoft's page share the same authentication session. The attacker's backend opened that session and was holding its state before the victim did anything at all. If the victim had signed in, the grant would have been approved, and the backend was positioned to collect the tokens it produced.

When the victim submits an email address in the phishing page, the browser posts the data to the attacker's end, not to Microsoft. The response is what sends the browser onward to

`login[.]microsoftonline[.]com/common/login`. That destination is picked server-side, and nothing in any URL the victim could have inspected points at Microsoft. All the calls described here were observed in the network capture. What happened between the attacker's server and Microsoft was never visible from the victim's side.

How Do We Know This Was a Device Code Flow?

The answer rests on the kit's own vocabulary. The endpoint is named `device/start` and the field is named `userCode`, and those names are the attacker's own, consistent with what has been observed in EvilTokens kit.

The decoded '`ctx`' is what actually settles it, because what inflates out of it is Microsoft's own record of the request its authorization server was handling.

Parameter	Value (this specific request)
`client_id`	`29d9ed98-a469-4536-ade2-f981bc1d605e` (Microsoft Authentication Broker)
grant/flow-type indicator	`OAuth2DeviceAuth`
`prompt`	`select_account`
`resource`	`https://graph.microsoft.com`
`tenant`	`common`
`claims`	`{"access_token":{"amr": {"essential":true,"values":["ngcmfa"]}}}`

The victim's consent screen shows the client is Microsoft Authentication Broker (Client ID: `29d9ed98-a469-4536-ade2-f981bc1d605e`) and the resource requested is `https://graph[.]microsoft[.]com`.

Together, the client and resource show what the attacker is after: tokens scoped to Microsoft Graph, Mail, Files and other account data.

In conclusion, three independent pieces of evidence points to:

1. The response's userCode value, nine uppercase characters with a 900-second lifetime, matches the shape RFC 8628 describes.

2. The decoded `'ctx'` names Microsoft Authentication Broker as the target client.

3. The grant type recorded in that same `'ctx' is OAuth2DeviceAuth`, Microsoft's own string for the Device Code Authentication.

Together, they confirm this is a genuine device code flow and the mitigations that work against real device code grants, blocking the flow, watching sign-in logs for the client ID, apply here too.

What Is Actually New Here?

Device code phishing is not new, and reverse proxy phishing kits are not new either. What's new here is the flow we observed: the kit opened the authorization grant, and the *user_code* was never shown to the victim.

The actions observed in this kit's code are similar to those in the EvilProxy kit, where a proxy sits between the victim and Microsoft and relays live session parameters onto the phishing page, as the victim interacts with it, nothing staged beforehand. Here, In contrast: the same parameter values were pre-staged in the phishing page before the victim ever interacted with it, something that was not observed in an EvilProxy-style relayed kit.

This attack is using a hybrid approach associating both AiTM attacks and Device code phishing attacks, which was not observed before.

What Do Defenders Need to Know?

- Restrict device code grants for non-essential users, as attackers are unable to alter this grant type. Additionally, track sign-in logs for device code authentication requests targeting the Microsoft Authentication Broker to detect these uncommon events.

- The transport layer is notable: Block UDP port 443 and clients fall back to TCP, which Cloudflare serves too, so the traffic still completes and your inspection stack can read it.

- Code-entry detections fail: The network log lacks "`microsoft[.]com/devicelogin`", "`login[.]microsoft[.]com/device`" navigations, and the "`?otc=`" parameter, leaving detection rules with nothing to flag.

- AiTM detections miss it for a different reason: AiTM detections look for a proxy relaying the victim's sign-in to Microsoft, such as sign-ins arriving from the relay or tokens replayed from elsewhere. Here the victim signs in directly on Microsoft's site, so that relay never exists for those detections to see.

We recommend keeping the old detections for AiTM and Device code phishing running as the same kit build still contains the classic branch that writes the code into the page and polls for approval.

While these security rules address previously documented scenarios, the critical insight lies in how the attacker orchestrated this chain. The initial email successfully passed domain authentication, and the link was delivered via a reputable email service provider. The user encountered a legitimate CAPTCHA and was ultimately directed to an authentic Microsoft sign-in page. Because users are trained to trust genuine Microsoft authentication portals, the victim made no discernible error. The attacker constructed an end-to-end chain where every user-facing step appeared completely legitimate.

The single most effective defense is evaluating the context before proceeding: "Does this sign-in prompt align with this user, device, and specific action?" An authentication attempt triggered by an unsolicited document approval request inherently lacks context. This contextual mismatch remains visible even when all technical indicators appear benign.

Indicators of Compromise (IOCs) to Block and Monitor

Indicator	Role in the chain
`polwanlo[.]vip`	First attacker-controlled domain. Took the click from the tracking link and redirected onward from `/.well-known/rr/`.
`*.merlekw[.]workers[.]dev`	Campaign router. Carried the campaign token in `?_k=` and issued the victim's transfer token for the next hop.
`*.maxjln[.]workers[.]dev`	Turnstile challenge stage and the kit page itself. The kit path only becomes reachable once Turnstile issues a pass token.
`c30d9e770432bc4-id-gate.bandgdruback1[.]workers[.]dev`	Backend. Served `/api/device/start`, the relay page at `/link/<token>`, and the email submission endpoint.

The domains rotate. These fingerprints are the more durable half:

Fingerprint	What it is
`POST /api/device/start` with an `X-Worker-Token` header	The call that opens the device authorization grant.
`/link/` followed by 32 hex characters	The relay page path.
The JSON fields `autoRelay`, `relayMode`, `relayToken` and `lureType`	The kit's own response contract. `relayMode` is the field that selects code suppression.
A sessionId shaped `ses_<unix_ms>_<8 hex>`	Server-generated session identifier.
`29d9ed98-a469-4536-ade2-f981bc1d605e`	Microsoft Authentication Broker. **A first-party Microsoft client. Do not block it.** Hunt for it paired with a device code grant.

Where KnowBe4 Can Help

Although this email bypasses email gateway controls and avoids triggering AiTM or FIDO2 detection mechanisms, multiple security layers still apply. Rather than relying on session replay tracking or domain reputation, protection depends on evaluating message content and user behavior.

KnowBe4 Defend: Real-Time Behavioral AI

The relevant layer here is content, not sender history, because SPF, DKIM, and DMARC all pass on this message, so nothing at the reputation layer has a reason to flag it. Defend's behavioral AI and natural language processing read the email itself rather than checking domain history, and the lure text, an unsolicited document-approval request running on an artificial deadline, is the kind of urgent, action-forcing language that analysis is built to flag, independent of how clean the sending domain looks.

Context-Aware Warning Banners

Defend adds a warning banner directly into the email before the recipient opens it, in this case the kind built for urgent action requests, ahead of the recipient doing anything with it. The banner acts at delivery, before a click, provided the classifier reads this specific lure pattern as risky in the first place.

Topics: Phishing | Multi-Factor Authentication (MFA) | KnowBe4 Threat Lab | KnowBe4 Defend | Account Takeover

See KnowBe4 Defend™ in Action

Learn how Defend™ strategically enhances Microsoft 365's native security to catch the threats Secure Email Gateways (SEGs) miss.

Request a Demo

Secure the Digital Workforce: Human + AI

KnowBe4 empowers the modern workforce to make smarter security decisions every day. Trusted by more than 70,000 organizations worldwide, KnowBe4 is the pioneer of digital workforce security, securing both AI agents and humans. The KnowBe4 Platform provides attack simulation and training, email and collaboration security, and agent security powered by AIDA (Artificial Intelligence Defense Agents) and a proprietary Risk Score. The platform leverages 15 years of behavioral data to combat advanced threats including social engineering, prompt injection, and shadow AI. By securing humans and agents, KnowBe4 leads the industry in workforce trust and defense.

knowbe4 THREAT LAB

ABOUT KNOWBE4 THREAT LAB

KnowBe4 Threat Lab is a research and analysis endeavor by seasoned cybersecurity professionals that discovers and investigates