ALLURE SECURITY

SECURITY RESEARCH

One Token to Rule Them All: Clustering a Payroll/Benefits AiTM Campaign

Four alerts came in over about seven minutes. Four different domains, four different Cloudflare edge IPs, resulting in four resolved takedowns. At first glance they may have looked like four separate phishing pages. Then we read the URLs and saw the same long, ugly query parameter riding along on every one of them, byte for byte identical.

A per-victim token is supposed to be unique. This one was identical across all four domains, byte for byte, and it wasn't just the base64 blob it looked like. It was signed. Following it linked the four domains into one operation, surfaced a second campaign, exposed the operator's own name for the job, and placed the kit precisely against this year's documented kits.

We'll show our work as we go. What follows will come in the order we found it, how we interpreted it, and what you can do about it.

One token, the short version:

- **A reused token, not shared infrastructure, tied the cluster together.** Four lure domains carried the same per-victim `ht` token byte-for-byte.
- **The benefits administration brand is bait, but Microsoft 365 is the real target.** Every path ends on a Microsoft sign-in served through an adversary-in-the-middle relay that captures the session token minted after MFA, so push, SMS, and TOTP don't stop it. The real prize is the mailbox, files, Teams, and every SSO-linked app behind that identity, not a benefits-portal password.
- **Who's exposed:** any Microsoft 365 organization whose staff can be lured with benefits, compensation, or HR themes, and any payroll/benefits provider whose brand is being baked straight into the lure domain.
- **What we add to the public record:** the first breakdown of the token's HMAC signature (what it is, what it prevents, namely editing and replaying links, and how to hunt on its shape), plus a self-dating `campaign_id`, a leaked operator label `Zoominfo_HR`, a variant that pushes the impersonated brand into the apex domain through a new registrar (NICENIC), and the use of IPv6.
- **What to do now:** require phishing-resistant MFA (FIDO2/WebAuthn, passkeys); treat any benefits/HR/compensation login that ends on a Microsoft page as a potential account takeover, not only a branding nuisance, and hunt on the token shape: `^eyJ[A-Za-z0-9_-]+\.[a-f0-9]{64}$` plus the `/compensation/auth/<hex>` path, both of which outlive any single two-hour, IP-pinned link.

How the cluster surfaced

Our research team tracks campaigns, not just individual pages. A phishing domain on its own is a data point. One that shares its kit fingerprint with a cluster of others is a campaign. The grouping work is handled by our internal correlation and campaign-mapping engine, which clusters detections by shared signals across a tenant and across that tenant's industry peers. The signals it weighs include registrar, certificate shape, hosting fingerprint, URL path templates, and kit-internal artifacts, among others. We do this because the operator isn't running one page, they're running a campaign, and you can't see the shape of something by looking at its pieces in isolation.

In this case the strongest signal was not the infrastructure. It was the kit's own tracking token, reused without modification across all four domains in the cluster.

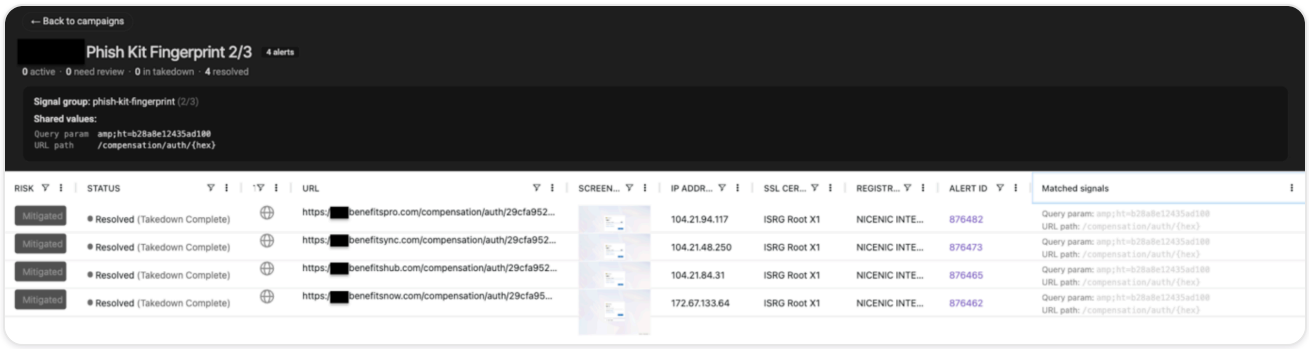

Figure 1. ThreatGraph cluster view. *Four detections grouped under the phish-kit-fingerprint signal group, all ultimately resolved via successful takedown.*

Reading the URL

Every URL in the cluster carried the same path and the same three query parameters.

```
/compensation/auth/29cfa952?s=3&ht=<token>&sso_reload=true
```

The path, `/compensation/auth/`, is the benefits-themed lure path. Each of the three parameters that follow it have a specific role.

The `s` value is a stage counter. It directly mirrors the `hop_count` we will find inside the token in a moment. This link is `s=3`, and the decoded payload lists a `hop_count` of 3. Across other samples we have analyzed but are not covering here, `s=1` aligned with a `hop_count` of 1 and `s=2` with a `hop_count` of 2. So `s` tracks the victim's position in the multi-hop redirect chain,

surfaced in the URL so the front end knows which part of the lure to render before it ever unpacks the token.

The `sso_reload=true` parameter shows up on every URL in the cluster. We cannot confirm its server-side function from the outside. The name suggests it nudges the Microsoft SSO reload behavior at the hop, but we are treating it as a stable part of the URL fingerprint rather than claiming to know what it does definitively.

That leaves `ht`, the parameter where the actual interesting part lives and is worth unpacking on its own.

Please sign here

At a glance `ht` looks like plain base64. It is not, or at least that's not all it is. The value comes in two parts split by a single period.

```
ht = <base64url-payload> . <64 hex chars>
```

The first part is base64 encoded JSON. The second part, `f1d225…ecf38a`, is exactly 64 hexadecimal characters, which is 256 bits. That is the size of a SHA-256 output, and in this position it is almost certainly an HMAC-SHA256 of the payload signed with a server-side secret. The structure is payload-dot-signature, JSON-web-token-shaped (JWT), but hand-rolled rather than a standards-compliant JWT.

Decoded, the payload is the kit's per-victim link object.


```
{
  "id": "9dd7a1e42ec839450c84b29bf6f3525e",
  "type": "hop",
  "campaign_id": "camp_69dc2dd7e19da",
  "hop_template": "benefits",
  "success_template": "benefits",
  "created": 1776103563,
  "expires": 1776110763,
  "ip": "<redacted IPv6 bound to the targeted recipient>",
  "max_uses": 100000000,
  "uses": 0,
  "hop_count": 3,
  "session": "sll44ur89timsgn7hfgl2e4de6"
}
```

The signature matters for two practical reasons. First, you cannot tamper with the token. The server validates the MAC, so editing the decoded JSON and re-encoding it, whether to swap the template, change the pinned `ip`, or push out `expires`, fails validation unless you hold the operator's secret. The token is integrity-protected rather than merely opaque or encrypted.

Second, that two-part shape (`token . hash`) is a high-fidelity hunt signature. A query parameter whose value is base64 starting in `eyJ`, followed by a dot and 64 hex characters, is distinctive and low false-positive. More on that later.

As for what `ht` stands for, our read is "hop token". The payload's `type` is literally `hop`, it carries `hop_count` and `hop_template`, and the value is a full signed descriptor rather than a bare identifier. It is inferred from the structure and not based on kit source code.

What the token actually controls

The decoded token payload is a per-victim access gate, and three of its fields do the enforcing.

The first is a hard-coded two-hour TTL. The `created` field is a Unix timestamp in seconds since the epoch (1 January 1970, UTC), so `1776103563` translates to (2026-04-13 18:06 UTC) and the `expires` timestamp (20:06 UTC) sit exactly 7,200 seconds apart (2hrs), so the links are short-lived by design. The second is IP pinning. The `ip` field binds the link to a single address, and a client that does not match, a sandbox or a rescan from a research IP, gets dead-ended. We have redacted that value because it is bound to the targeted recipient. The third is the MAC, the integrity check described above.

`max_uses` is `100000000`, a hundred million, with `uses` sitting at zero. That is not a usage cap in any meaningful sense. At a hundred million it is effectively uncapped, and the single-use throttling you might assume from a field named `max_uses` is not doing much of anything. The TTL, the IP pin, and the MAC are.

A last tell sits in the `campaign_id`. `camp_69dc2dd7e19da` looks like a PHP `uniqid()` value, whose leading eight hex characters encode a creation timestamp. `0x69dc2dd7` decodes to roughly 18 hours before this link's own `created` time, which is consistent with the campaign being spun up first and the per-victim hop tokens minted later. That is a small detail, but it has a bearing on what kind of kit this is, which we come back to below.

OK, but why is the token the same

Here is the cluster laid out. Same decoded `id`, same `campaign_id`, same path template, sitting on four different domains served from four different Cloudflare edge IPs.

Detection	Domain	Edge IP	ht.id	campaign_id
1	███benefitsnow[.]com	172.67.133.64	9dd7a1…3525e	camp_69dc2dd7e19da
2	███benefitshub[.]com	104.21.84.31	9dd7a1…3525e	camp_69dc2dd7e19da
3	███benefitsync[.]com	104.21.48.250	9dd7a1…3525e	camp_69dc2dd7e19da
4	███benefitspro[.]com	104.21.94.117	9dd7a1…3525e	camp_69dc2dd7e19da

A per-victim hop token is supposed to be unique. This one is verbatim across four registered domains. The simplest interpretation is that these are four interchangeable front doors for one campaign rather than four separate campaigns, with the operator either seeding the same minted link across a domain pool or generating links from a single template and rotating the hostname. The Cloudflare IPs differ only because Cloudflare's edge load-balances them. The origin behind them is the same kit.

That pool is redundancy by design. With four interchangeable hostnames pointing at one origin, taking down a single domain dents the campaign without ending it, and the operator can stand up a replacement against the same kit without touching the back end. It also raises the value of clustering: knock down one and you have learned little, but tie the four together and a takedown can target the whole set at once.

And it is deterministic, not inferred. The kit handed us its own internal identifier, and the operator used it as a shared key across the campaign run.

The victim POV

Every one of these detections is tagged as a payroll/benefits brand-impersonation alert. The domains embed the brand string and the lure is benefits-themed. The screenshot captured on each page tells a different story. It is a Microsoft sign-in page.

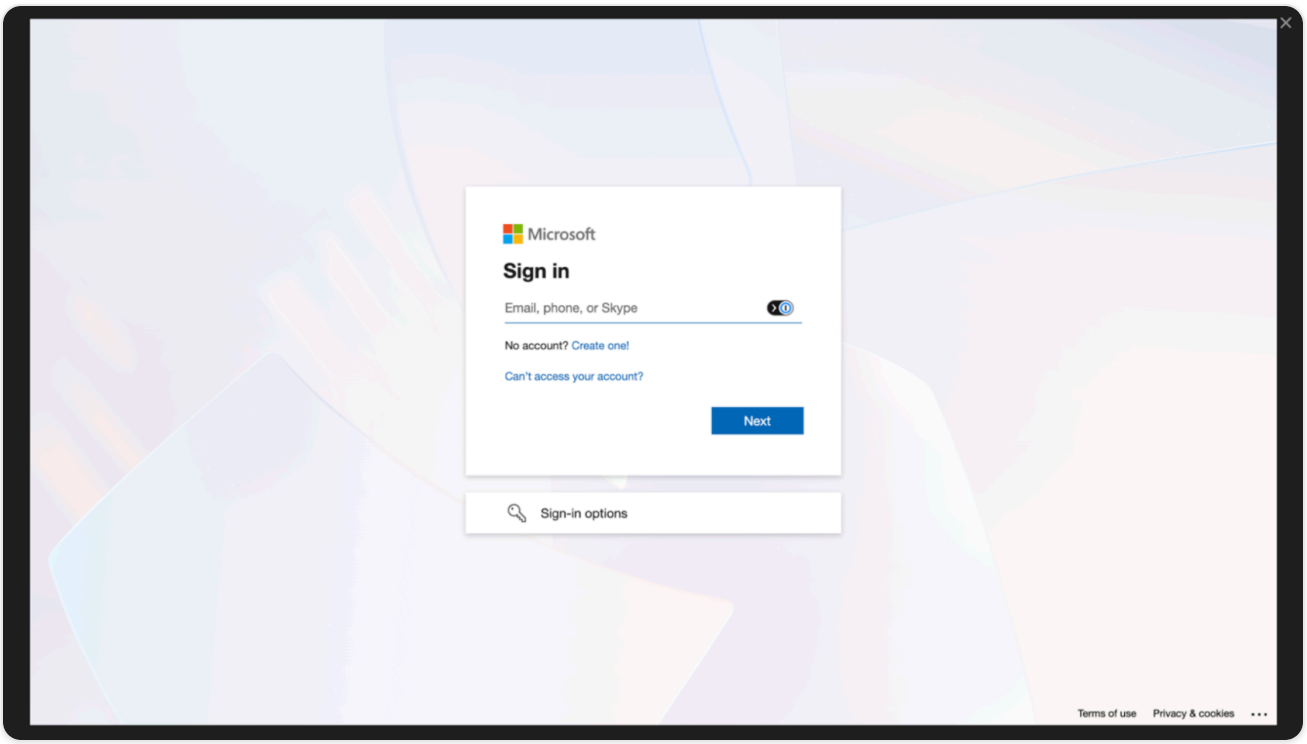

Figure 2. Captured render. *The Microsoft "Sign in" page served at the end of the redirect chain on [redacted]benefitspro[.]com.*

The benefits branding is the initial bait. The harvesting page is Microsoft 365, and that twist is the whole point of the operation. A lure email points to a benefits-themed domain that looks official, brand in the hostname and a `.com` TLD. The visitor lands on the hop page, rendered from `hop_template` benefits, a brief on-theme splash, often a captcha to confirm you are human.

The kit then redirects to a Microsoft 365 login intercept running as an adversary-in-the-middle (AiTM) relay, which captures the entered username, password, and session token minted after MFA. Last, the victim sees the `success_template`, also benefits, so the experience ends on a plausible "you are all set" note. No glaring tells to tip you off along the way.

A single stolen benefits-portal password has limited worth. A live Microsoft 365 session token is worth the victim's mailbox, files, Teams, and any SSO-linked applications hanging off that identity, very possibly including the real payroll/benefits app the lure was imitating. The benefits brand gets your attention, but M365 is the primary target. Because the captured

The brand the victim is initially lured with and the brand that ultimately robs them are not the same, and that is worth calling out in any alert from a cluster like this. Categorized as benefits-portal lookalikes, these may read as a branding nuisance on the surface. Categorized as Microsoft 365 session theft, they read as an account-takeover threat. Same domains, but a very different impact.

The infrastructure

The four domains share a consistent build. All are registered through NICENIC INTERNATIONAL GROUP CO., LIMITED (IANA 3765), all registered on 2026-04-12 for a single year. Each carries a Let's Encrypt certificate issued the day after registration, with wildcard SANs covering `*.<domain>` alongside the apex, under issuers E7 and E8. All sit behind Cloudflare, with name servers on the `*.NS.CLOUDFLARE.COM` pattern.

The wildcard certificate is a detail worth lingering on. It lets the operator stand up arbitrary subdomains, `secure.` and `login.` and the like, under each registered domain without minting new certificates. If you are hunting this, do not strictly anchor on the apex domains. Leave room for the subdomains, because the operator already has.

But wait, there's more

Pivoting on the kit fingerprint surfaced a second run from the same kit, this time on `[redacted]-benefits-hub[.]com` and `[redacted]benefitspeak[.]com`, carrying a different token.


```
{
  "type": "campaign",
  "campaign_id": "camp_69ed67e9320f0",
  "campaign_name": "Zoominfo_HR",
  "hop_template": "benefits",
  "success_template": "benefits"
}
```


The hop_template and success_template values match, so it is the same kit code, but the

Then a small gift. `campaign_name` is set to `Zoominfo_HR`, a field the kit doesn't need to function. It's the operator's own internal label for the run, the kind of metadata that normally never leaves their console. What it points to is open to interpretation: targets sourced from

ZoomInfo, an HR-themed lure, or simply the operator's shorthand for this batch. We can't confirm which from the string alone. What we can say is that the operator names runs by theme or source, which is a small window into how they organize their work, and more useful for anticipating the next campaign than another row of IOCs.

What kind of kit is this

Put the token internals together and a sharper picture emerges than just "an AiTM phishing page."

This object is the redirector tier of a templated phishing-as-a-service front end, the hop layer that mints per-victim tokens and decides which themed page to render. It sits ahead of any credential capture and is not the live-proxy plumbing itself. Three things point that way.

The `type` is `hop`, with `hop_count`, `hop_template`, and `success_template` alongside it, which is staging-tier behavior, a redirector choosing which template to show at the hop and after capture. The benefits value is a lure theme drawn from a template library, the open-enrollment and HR bait, which is library behavior rather than live-proxy plumbing. The short TTL, the IP pinning, and the single-issue links are textbook anti-analysis controls built to dead-end sandboxes and rescans. That is the same category of gated-access Datadog documented in an unrelated 1Phish kit, which validates victims through an `/api/validate-access` endpoint before serving content. Different kit, different target (1Password), but the same "validate before you serve" approach.

And the PHP fingerprints run throughout, from the `uniqid()`-shaped `campaign_id` to a `session` value that is 26 lowercase-alphanumeric characters, the default shape of a PHP `PHPSESSID`. Taken all together this can be categorized as a PHP-based templated redirector in the Dadsec, Tykit, and Greatness lineage rather than EvilProxy or an indirect-proxy operation, which brings us to the attribution question.

Where we've seen this kit, and where it doesn't fit

This kit family has been documented before, and we want to point to that work rather than imply we got here first.

Okta Threat Intelligence documented it on 21 October 2025 in "Phishing campaigns use 'Employee Benefits' lure to intercept Microsoft and Okta logins" (Houssem Eddine Bordjiba),

tracking a campaign they label O-UNC-037. Their writeup describes the same `ht` token with the same `type` hop, `campaign_id` , and `hop_template` schema, the benefits and HR lure, the Microsoft AiTM stage, and a second stage that intercepts the Microsoft federation response to redirect Okta-federated victims to a fake Okta page. The path template in our cluster, `/compensation/auth/` , appears in their list of the kit's lure paths.

Datadog Security Labs published a complementary analysis on 10 December 2025, updated in March 2026, "Investigating an adversary-in-the-middle phishing campaign targeting Microsoft 365 and Okta users" (Martin McCloskey, Christophe Tafani-Dereeper, and Julie Agnes Sparks). Their post works the defender-log angle, covering the client-side `inject.js` cookie stealer and its list of critical Okta session cookies, the `okta-secure[.]io` style proxy pages, initial access through compromised Salesforce Marketing Cloud and Amazon SES senders and password-protected PDF attachments, and detection queries for Okta and Microsoft 365 logs. Their decoded token matches the schema we observed.

Both teams came at this from the identity-provider and SIEM-log sides, answering what the kit looks like in authentication telemetry. Both are worth reading in full. The credential-stealer and federation-interception internals they document are the inside of the same machine we are describing from the outside.

What this grammar does NOT match is also useful to highlight. We checked the current crop of named Microsoft 365 phishing kits, and the signed `ht` hop-token grammar, with its `hop_template` and `success_template` , its `s` stage counter, and its `sso_reload` flag, does not line up with EvilProxy, Tycoon 2FA, or Storm-1167, nor with the newer 2026 kits.

EvilTokens, documented by Sekoia in April 2026, is built differently. Its pages are an empty div plus a script that fetches base64 content and decrypts it in the browser with AES-GCM through the Web Crypto API, and it runs off Microsoft device-code authorization. There is no signed URL token of this shape anywhere in it.

Kali365, the subject of an FBI public service announcement on 21 May 2026, is an OAuth and device-code token-capture platform. It abuses the legitimate device-authorization flow instead of fronting a templated redirector with HMAC-signed hop tokens.

Our honest read is that this is a custom or not-yet-catalogued PHP templated redirector. It is a notch more engineered than the average commodity kit. Most simply base64 their tracking blob. This one signs the hop token with an HMAC and carries PHP fingerprints throughout. It sits in the Dadsec and Tykit feature space, but we will not pin it to a named kit on this evidence alone. If anyone reading has source-code from a live instance, that is what would settle both the family question and the meaning of that last `sso_reload` parameter.

What's different in our cluster

We see this from the brand-protection and external-attack-surface side, watching the lure domains as they stand up on the internet, before a single employee at the targeted brand has to click anything. From that angle, a few things in this cluster read as a variant rather than a carbon copy of what Okta and Datadog described.

The most visible difference is the naming. The earlier writeups catalogued largely generic hostnames, things like `benefitsemployeeaccess[.]com`, `mybenefits-portal[.]com`, and `employee-hr-portal[.]com`. Our cluster embeds the impersonated payroll/benefits provider's brand directly into the registered domain, in the `[redacted]benefits*` form, which sharpens the lure into something provider-specific. Datadog's March 2026 IOCs show early co-branding through subdomains like `511tactical.totalbenefitsportal[.]com`. Our cluster pushes the brand all the way into the apex domain.

We also took the signed token further than the prior posts did. Okta's published sample carried the same trailing 64-hex suffix, but neither writeup drew out what it is, an HMAC over the payload, what it prevents, which is editing and replaying a token, or how to hunt on its shape. That analysis, along with the `s` and `hop_count` correspondence and the hop-token reading of `ht`, is part of what we are adding to the record.

The registration channel differs too. Datadog noted that most of their domains came through NameSilo. Ours run through NICENIC (IANA 3765), the same kit reaching the internet through a different registrar, consistent with either a different operator buying the same kit or the same operator rotating registrars.

A couple of smaller markers separate the samples. The Okta and Datadog tokens carried `success_template` `benefits_error`, where ours carries `benefits`, a minor config change but the kind that helps tell operators or kit versions apart. The `max_uses` value also moved, from `10000000` in the prior samples to `100000000` in ours. Neither value functions as a real cap, so this is not a change in how access is controlled, only a config-value fingerprint that may track kit version or operator. The pinned `ip` in our token is IPv6 where prior samples showed IPv4, a small but concrete difference in the captured artifact.

There's a smaller forensic bonus in the `campaign_id` itself. `camp_69dc2dd7e19da` is a PHP `uniqid()` value, and the leading eight hex characters of a `uniqid()` are a Unix timestamp. That makes every campaign id from this kit self-dating. `0x69dc2dd7` decodes to roughly 18 hours before the token's `created` time, and the second campaign's id (`camp_69ed67e9320f0`) decodes later still, so the two runs can be ordered in time from the ids alone. The caveat is that `uniqid()` timestamps are trivially forgeable, so this holds only as long as the operator wasn't fabricating them, which kit authors rarely bother to do.

The methodological difference is the one we care about most. Rather than grouping on infrastructure or matching kit source, we clustered on the operator's own reuse of an identical hop token across a domain pool, a deterministic link the operator created for us by being lazy with a value that was supposed to be unique. That reuse, plus the leaked `campaign_name`, is what our systems lifted out of the background of routine alerts.

None of this contradicts the prior work. It extends it. The kit is still in active development and active use, it is being run as more than one operation, and at least one operator has moved toward provider-specific brand impersonation through a new registrar.

A detection you can build today

The URL is highly fingerprintable, and the strongest single signature is the shape of the `ht` value.

```
ht value regex:  ^eyJ[A-Za-z0-9_-]+\.[a-f0-9]{64}$
```

That is base64 JSON, where the `eyJ` prefix is `{"` encoded, followed by a dot and a 64-hex MAC. Pair it with the optional `s=<int>` stage counter and an `sso_reload=` parameter and you have narrowed it sharply. Decode the payload, confirm the keys `type":"hop"`, `hop_template`, `success_template`, and `hop_count`, and you are at production grade signature.

One operational caveat matters. Because the live token is IP-pinned and good for only two hours, you cannot detonate a stale sample and reach the live page. The server dead-ends you the moment your IP or the clock does not match, so retro-hunting a captured URL will not render the page. Pivot instead on the token shape and the lookalike-domain path structure, `/compensation/auth/<hex>`, which persist whether or not any single link is still live.

Response

These four detections did not sit in a dashboard. The managed workflow ran end to end, from SOC review through takedown approval to blocking requests submitted to APWG eCX, CleanDNS, Google Web Risk, OpenPhish, and Spamhaus, with Spamhaus blocking confirmed and takedowns completed across the cluster. The same external vantage point that makes brand impersonation visible is the one that lets the lure infrastructure come down before the emails even land.

Indicators of compromise

Brand string redacted as ████. Defanged.

LURE DOMAINS (CAMPAIGN CAMP_69DC2DD7E19DA)

- ███benefitsnow[.]com
- ███benefitsync[.]com
- ███benefitshub[.]com
- ███benefitspro[.]com

LURE DOMAINS (CAMPAIGN CAMP_69ED67E9320F0, ZOOMINFO_HR)

- ███-benefits-hub[.]com
- ███benefitspeak[.]com

Kit and campaign artifacts

URL path template	`/compensation/auth/<hex>` (observed: `/compensation/auth/29cfa952`)
URL parameters	`s=<stage>` and `sso_reload=true`
ht token shape	`^eyJ[A-Za-z0-9_-]+\.[a-f0-9]{64}$` (base64url payload + HMAC-SHA256)
ht token kit id	`9dd7a1e42ec839450c84b29bf6f3525e`
campaign_id	`camp_69dc2dd7e19da`
campaign_id	`camp_69ed67e9320f0`
campaign_name	`Zoominfo_HR`
hop_template	`benefits`
success_template	`benefits` (vs. `benefits_error` in prior public samples)
session shape	PHPSESSID-style, 26 lowercase-alphanumeric chars

Infrastructure pattern

Registrar	NICENIC INTERNATIONAL GROUP CO., LIMITED (IANA 3765)
Certs	Let's Encrypt, wildcard SAN (`*.<domain>`), issuers E7 / E8
Fronting	Cloudflare (NS: `*.NS.CLOUDFLARE.COM`)
Edge IPs	`172.67.133.64, 104.21.84.31, 104.21.48.250, 104.21.94.117`
Domain age	~1 day at detection, 1-year registration

A note on the lookalikes

If you defend a Microsoft 365 estate, treat the benefits theme as misdirection. Any benefits, compensation, or HR-themed login that ends on a Microsoft page should be handled as an M365 session-theft attempt, which means putting phishing-resistant authenticators, FIDO2/WebAuthn and platform passkeys, on the accounts that matter. If you are a brand that gets impersonated this way, the domains tend to stand up days before they are used, so watching for brand-in-domain registrations on registrars like NICENIC, fronted by Cloudflare, carrying day-old wildcard Let's Encrypt certificates, buys you the head start.

We will keep tracking this kit. It is still being built.

Thanks to Okta Threat Intelligence and Datadog Security Labs, whose prior public analyses of this kit family are linked above and are well worth reading alongside this post. The EvilTokens (Sekoia), Kali365 (FBI), and 1Phish (Datadog) references are linked inline for contrast, different kits named here only to show what this one is not.