

FIELD NOTES FROM A PRESALES SOLUTIONS ARCHITECT

"We've Been Hacked" — No, That Is Cloudflare: Reading Edge Traffic Before You Blame the VM

A presales war story about a firewall full of "abnormal sessions", a team ready to block their own origin IP, and the three questions I now ask in Cloudflare before anyone calls a server compromised.

Daanial Farrukh · 28 October 2026 · <https://quabin.com/cloudflare-traffic-not-vm-compromise/>

The call came on a weekend.

"The VM is compromised. The firewall is showing abnormal sessions all over it."

The server was a customer's cPanel box — WordPress sites and email on one VM, with Cloudflare in front of the websites on the free plan.

The plan was already moving.

"We're running an antivirus scan now. And we want to block the VM's IP on the firewall until it's clean."

I asked one question.

"Where are the sessions coming from?"

Someone read a few source addresses off the firewall view. They all started with 172.64, 172.68, 162.158.

"Those are Cloudflare."

Silence.

"So... Cloudflare is attacking us?"

The Trap, Named

That's the trap. **Inbound traffic from Cloudflare to your origin is not an attack on your VM. It is your website working.** A busy, angry firewall view looks exactly like a breach to someone who does not know which way the arrows point.

And blocking the origin IP does not stop anything. It just takes every site on the server offline — the attacker's job, done for free.

What You Are Actually Looking At

When a site sits behind Cloudflare's proxy, nobody on the internet talks to your server directly. The path is:

Visitor → Cloudflare edge → Origin VM

Every request your origin sees on port 443 arrives from a Cloudflare edge IP. Cloudflare publishes those ranges — 173.245.48.0/20, 103.21.244.0/22, 104.16.0.0/13, 172.64.0.0/13, 162.158.0.0/15 and a handful more. So when a firewall lists thousands of inbound :443 sessions from 172.64.0.0/13 and 162.158.0.0/15, it is showing you **reverse-proxy traffic**, nothing else.

A compromised VM is the source of traffic, not the destination. The signs point outward:

- the VM opening connections to addresses nobody configured — a command-and-control server, a mining pool
- spam or scans leaving the server from processes that should not send anything
- a login that succeeded from an address or a tool that should never have one

Inbound sessions from Cloudflare are the opposite direction. They tell you about load, not about a breach.

This mix-up is common in three places:

- teams where the network side and the web side are run by different people
- monitoring that alerts on session count without knowing what sits behind the proxy
- any shop that has added Cloudflare recently and never updated its "normal" baseline

So What Was It?

The sessions were real, and there were a lot of them. I opened Cloudflare's Security Analytics for the zone and read the last 24 hours.

- **About 22.7 million requests** in 24 hours, for sites that normally see a tiny fraction of that.
- **About 22.7 million GET and about 7,200 POST.** So this was not a password attack on wp-login.php or xmlrpc.php — those are POSTs.
- **Two user-agent strings carried almost all of it**, roughly 50/50: one Firefox on Windows, one Firefox on iOS. Real Chrome — the browser most real visitors use — was about 0.25 million.
- **Cloudflare mitigated about 1.18 million (~5%)** and passed **about 20 million (~88%)** to the origin as dynamic, uncached requests.

That is a distributed layer-7 GET flood from a botnet pretending to be two kinds of Firefox. Every one of those 20 million requests reached the VM through Cloudflare. That is what lit up the firewall.

Here's the part that trips people up. **Nothing on the VM was wrong.** The antivirus scan would find nothing because there was nothing to find. The VM was doing its job under a flood.

Why the Usual Check Misses It

The first place most people look in Cloudflare is **Security → Events**. On this zone it was nearly empty. A few blocks, a few challenges — nothing that matched 22 million requests.

That empty log is the second trap. On the free plan, a flood that looks like real browser traffic does not show up as `ddos_block` or `managed-challenge` events. Cloudflare did not decide it was an attack, so it did not log an action. **An empty events log means Cloudflare did not act. It does not mean nothing happened.** Ask a dashboard "was there a DDoS?" and it will look at the events log and say no.

You judge a flood by **volume and shape**, not by the events log:

- request count against the normal baseline
- the GET/POST split
- the user-agent mix
- the top source ASNs

All four live in **Security** → **Analytics**, not in Events.

How to Actually Tell — and Fix It

Step 1: Confirm the sources are Cloudflare

Before anything else, check the source addresses the firewall is worried about against Cloudflare's published list:

```
curl -s https://www.cloudflare.com/ips-v4
```

If the "abnormal" sources sit inside those ranges, you are looking at proxied traffic. Stop the VM-blocking plan here.

Step 2: Read the shape of the traffic in Cloudflare

Cloudflare → your zone → **Security** → **Analytics**. Set the window to the last 24 hours and read four panels in this order:

- 1 **Total requests** — compare with a normal day.
- 2 **HTTP methods** — mostly GET means a page flood; a POST spike on login paths means a credential attack.
- 3 **User agents / browsers** — real traffic is spread across many browsers. One or two identical strings carrying most of it is a botnet template.
- 4 **Mitigated vs served by origin** — how much got through to you.

Plain takeaway: if requests are huge, mostly GET, and two user-agents carry the load, it is a flood. It is not a breach.

Step 3: Turn on the free plan's strongest lever

Cloudflare → **Security** → **Settings** → Security Level → **I'm Under Attack**.

This puts a JavaScript challenge in front of every visitor. A real browser passes it in a few seconds without the user doing anything. A raw GET-flood script cannot run JavaScript, so it never reaches the origin. On the free plan, this single switch does more than anything else.

Optionally, add a custom rule with **Managed Challenge** for the exact user-agent strings the botnet is using, so you can dial the global level back down later without letting the template through.

Step 4: Lock the origin to Cloudflare

A flood that cannot pass Cloudflare will try to go around it. Allow port 443 on the origin only from Cloudflare's ranges, at the firewall in front of the VM or on the VM itself. Then the botnet has no direct path. This is the one change on the network side that actually helps — the opposite of blocking the origin IP.

Step 5: Verify — this step is where most tutorials stop and where mine starts

Back in **Security** → **Analytics**, the "served by origin" line should fall to a small fraction of the total, and the challenged count should rise to match the flood. On the origin, the number of established sessions on :443 should drop:

```
ss -Htn state established '( sport = :443 )' | wc -l
```

If origin traffic does not drop, something reaches the server without passing Cloudflare — a DNS-only record that points at the same IP, or port 443 still open to the world. That is not a failed fix. It is the verification telling you where the gap is.

One more thing that confused the team: requests to wp-cron.php coming **from the origin's own IP**. That is WordPress calling its own scheduler through the public URL. It is the site talking to itself, not an attacker.

What a Real Compromise Looked Like — on the Same Server

I want to be honest here: this same server did have a real compromise, found a few days later. It is the best proof that the two look nothing alike.

First, there was probing. A scanner with an empty user-agent asked a backup subdomain for a list of common web-shell file names. Probing is not a breach. The question is whether any of those requests got a real answer. One grep on the access log decides it:

```
grep -Ei "sh3ll|ah25|xwpg" access_log | grep " 200 "
```

Nothing means the scanner knocked and nobody answered. A 200 means a file with that name existed and ran — then you investigate that file.

The real entry, when it came, was quiet. **One successful login to the cPanel account, from a script user-agent (python-requests), followed by a password change on that account.** No flood, no firewall alarm. It was found in the cPanel login and access logs, not in Cloudflare and not on the firewall.

And during the cleanup, two process names in top looked like malware. Each one was checked before anyone called it:

```
ls -l /proc/<PID>/exe
```

One pointed to a mail server's own helper binary — legitimate, just stuck at full CPU from brute-force load on the mailboxes, cleared by restarting the mail service. The other was an admin tool running from root's home directory. Neither was malware.

A breach is a quiet login and something leaving the server. A flood is loud traffic arriving through the proxy. Mixing them up costs you both ways — you block your own sites for a flood, and you miss the

one login that mattered.

The Bigger Lesson — Five Checks Before You Say "Compromised"

- 1 **Which direction is the traffic?** Inbound from Cloudflare ranges is your site. Outbound to unknown addresses is the alarm. `ss -tnp` state established shows both, with the process.
- 2 **Is there a successful login you cannot explain?** For cPanel, read `/usr/local/cpanel/logs/login_log`. For SSH, `grep Accepted /var/log/auth.log`.
- 3 **Did any probe get a 200?** Grep the access log for the probed paths and the status code.
- 4 **Is the scary process really what its name says?** `ls -l /proc/<PID>/exe` before you kill or report it.
- 5 **Is the edge absorbing the load?** Security → Analytics: mitigated vs served by origin.

If all five are clean, you have a load problem, not a security problem. Fix it at the edge.

What I Tell Every Customer Now

When a team says "the VM is hacked", I ask one question before anything is blocked or scanned:

"Show me one connection that **started** from the server."

If the answer is "there isn't one, they're all coming in from Cloudflare", we are dealing with traffic, and the fix is in Cloudflare. If the answer is "yes, it's talking to an address we don't know", we're not done yet — and now we look at the VM.

I write about the questions I ask in real customer rooms — the ones that change architectures and surface the problems no one else noticed. If you found this useful, drop your email below and I'll send the next one when it's ready.

FAQs

Why does my firewall show so many connections from Cloudflare IPs?

Because your site is behind Cloudflare's proxy. Every visitor's request reaches your server from a Cloudflare edge address, not from the visitor. Thousands of inbound sessions from ranges like 172.64.0.0/13 or 162.158.0.0/15 are normal reverse-proxy traffic, and a spike means more requests, not a breach.

How do I know if my server is compromised or just under a DDoS?

Look at direction. A compromised server starts connections outward — to a command server, a mining pool, or other mail servers with spam — or shows a login you cannot explain. A layer-7 flood is huge inbound request volume through the proxy. Check outbound connections and login logs on the server, and request volume in Cloudflare Security Analytics.

Why is the Cloudflare Security Events log empty during an attack?

Events only lists requests Cloudflare took an action on. On the free plan, a flood that imitates real browsers is often not classified as an attack, so it is passed to the origin without an event. Judge the attack by total requests, method split and user-agent mix in Security Analytics instead.

Does Cloudflare's I'm Under Attack mode stop a layer-7 GET flood?

It stops most simple ones. It shows a JavaScript challenge that real browsers pass automatically and plain request scripts cannot. Combine it with a firewall rule that allows port 443 on the origin only from Cloudflare's ranges, so the flood cannot go around the proxy.

Should I block my origin IP during an attack?

No. Blocking the origin IP takes every site on the server offline, which is what the attacker wanted. Block or challenge at Cloudflare, and restrict the origin to Cloudflare's ranges so nothing reaches it directly.

Why do I see wp-cron.php requests from my own server's IP?

That is WordPress triggering its own scheduled tasks by requesting wp-cron.php through the public URL. The source is your origin because the server is calling itself. It is expected and not an attack.

Read online: <https://quabin.com/cloudflare-traffic-not-vm-compromise/>