A bomb is not a control

A static site is a rare place where one class of wrong guess can be made expensive. A zip bomb taxes the scanner that doesn't look before it unpacks — but the bomb is the theatrical half. The ban is the control, and getting the ban right is most of the work.

Matthew J. Harmon

2026-09-11

The central claim of this site is that an attacking agent is allowed to be wrong for free. It can walk a wordlist of ten thousand URLs, miss on nine thousand nine hundred and ninety, and the misses cost it nothing, because a miss is just the next line of the list. That is the shape of the whole trade, and it is why you cannot, in general, make the attacker's search expensive. Search is the thing these systems are good at.

But *in general* is carrying weight in that sentence. There are specific places where one wrong guess can be made to cost more than it returns, and a static site is one of them.

The site is boring, and that is the signal

Nothing here logs in. A static site publishes no forms, runs no PHP, keeps no admin panel, and has no `.env` to leak. So a request for `/wp-login.php` or `/.env` or `/api/v1/users` is not a visitor who took a wrong turn — there is no path on the site that leads there. It is an automated scanner reading a wordlist of things that are valuable *when they exist*, against a site where none of them do.

The wordlist is the confession. Ordinary readership never produces that sequence; a generic vulnerability scanner produces little else. That makes it a high-confidence *event* — the classification is about as clean as this work gets — though the address behind the event is a separate and weaker question, one the rest of this piece keeps returning to. What makes the classification clean is worth being precise about. It is not that the request is unusual; it is that **you know exactly what your own site contains, and the attacker's tooling is guessing against a generic host that isn't there.** The signal is an artefact of the gap between what they assume you are and what you actually are.

What the bomb is

A zip bomb: forty-two kilobytes on the wire that unpack to five and a half gigabytes — roughly a hundred and thirty thousand to one. Its two hundred and fifty entries overlap their compressed data, so the file stays a tiny fraction of the size of everything it claims to hold. It is served with a `200` and `Content-Type: application/zip`, so it reads to a scanner as exactly the kind of leaked archive its wordlist was hunting for. One copy is held in memory and handed to every probe on every hostname.

The server never decompresses it — it ships the compressed bytes verbatim. That is the whole asymmetry: you spend forty-two kilobytes of RAM and a few microseconds; a scanner that unpacks what you sent spends five and a half gigabytes. For once, the wrong guess is expensive, and it is expensive for *them*.

The bomb is the weak half

The bomb only bites something that inflates the file without looking at it first. A careful client reads the `Content-Length` — forty-two kilobytes — and the declared type, and decides what to do before it commits any memory; it never pays. Any decompressor built this decade can cap output or stream against a ceiling. So the bomb is a bet that the scanner is as careless as the cheapest tooling on the shelf. A great deal of it is. But you cannot count on it, and you must not build the defense around it.

The defense is the ban. Every probe is logged with the client's address, and a few within a few minutes trigger a temporary, port-scoped firewall rule for that address. The ban does not care whether the attacker took the bait — it fires on the request, not the response. That is the half that actually shortens the engagement, because it stops the rest of the wordlist from ever running.

> The bomb is theatre that sometimes lands. The ban is the control. The temptation is to reverse them, because the bomb is the fun part — and the fun part is the part a competent adversary walks straight past.

What it costs to do it right

The unglamorous half is the real work, and none of it is the bomb.

You need the client's true address, and it is easy not to have it. A server behind a connection-level proxy sees every request arriving from the proxy — one internal address, identical for the whole internet. You cannot ban that. Recovering the real address means the proxy has to prepend it to each connection and the server has to trust that prefix — and trust it *only* from the proxy, because a header anyone can send is a header anyone can forge. The address is worth the trouble because completing the handshake vouches for it in a way a header cannot: the peer had to receive and answer real packets, so it can't be forged by the blind spoofing that fakes a client-supplied field. It is not the attacker's true or only address — a VPN exit, a relay, a proxy or a carrier NAT can sit in front of it — but it is the network source that actually spoke to you, which is what a firewall rule acts on regardless. Wiring that through was most of the effort, and the bomb was the least of it.

A log line decides who gets banned, and the log line is attacker-controlled. The requested path is written into the line the ban system reads. A path containing a newline can forge a *second* line naming any address the attacker likes — and get an innocent third party banned, or get you banned from your own site. The fix is to escape the field so a newline renders as two harmless characters and a quote cannot close its own field. It was tested by sending a path that forged a complete fake entry nominating a stray address; the stray address was never banned. The general lesson outlives this case: any pipeline that turns a log line into a firewall rule is a pipeline where hostile text writes firewall rules, and it has to be built like one.

The ban has to be narrow in two directions. Scoped to the web port, so it can never lock you out of the machine over SSH. And short in time, because the address behind a clean signal is not always a single culprit — it can be a carrier-grade NAT or an office gateway shared by a thousand blameless readers. An hour is enough to shed an automated sweep without punishing a building for a day.

The pattern list is a standing liability. `.php`, `/wp-*`, `.env` are safe forever on a site that will never serve them. But `/api` is a bet about the future: the day you stand up a service with a real `/api`, that rule starts banning its users. The exception has to be carved before the new thing ships, not after the first complaint.

The trap inside the trap

Two footnotes, both earned by getting it wrong first.

When I went to plant the decoy, I found a file already sitting in one site's directory named `wp-login.php` — and it was itself a forty-two-kilobyte zip bomb someone had left there months earlier for exactly this purpose. The scanner-bait had collided with real scanner-bait. A blanket rule that banned every request to that path would have banned visitors to a working decoy. Look at what is already in your own tree before you assume you know what a path means.

And the trap lives in one server, which not every request reaches. Some names are fronted by other layers that answer first, and a probe that never arrives is a probe the trap never sees. A defense is only as wide as the requests that actually reach the thing enforcing it — worth knowing, name by name, what answers first.

The shape of it

You cannot make search expensive. The attacker's ninety-nine free failures are, in the general case, free. But a static site is a place where one narrow class of failure can be made to cost — not because the defender is clever, but because the site is boring in a way the defender can describe exactly and the attacker's tooling cannot. The scan announces itself against a surface that has nothing it was scanning for.

And even here, the expensive part only lands on the guesser who leaps before it looks. Make the wrong guess expensive where you can; the bomb is a cheap bet that occasionally pays. But the guess was always going to be cheap for a careful adversary, and the durable move is the same one it has always been: notice the thing that has no legitimate reason to happen, and act on it while you still can — quietly, narrowly, and somewhere the attacker cannot reach.

This site serves that bomb. If you fetched `/wp-login.php` on your way in, you are holding a five-gigabyte souvenir and an hour to think it over.