monarx

CUSTOMER STORY

MONARX CUSTOMER STORY

Two minutes to cut off a live attacker

How Monarx contained an in-progress intrusion across a small agency's managed WordPress fleet.



TIME TO CUT OFF WRITE ACCESS

minutes from the attacker's first shell request to Monarx cutting off write access

WRITE ACCESS CUT	C2 DROPPER NEUTRALIZED	BLOCKED IN FIRST 3 DAYS
2 min 31 sec	Within seconds	7,800+ runtime events

SUMMARY

Weeks inside the fleet. Then one night it stopped.

A small independent web agency managing production WordPress infrastructure for several much larger organizations discovered that attackers had been inside its server fleet for weeks. The intrusion spanned multiple clusters and used webshells, botnet toolkits, SEO cloaking, hidden administrator accounts, and persistence in locations that survived every web-root cleanup.

The agency was mid-remediation when it approached Monarx. It sat well below Monarx's usual customer profile, and there was no commercial reason to prioritize it. Monarx helped anyway — agents went live across the fleet within roughly a day, before a contract existed.

Days later, an attacker returned with a live webshell kit. Monarx was the only automated control that acted during the session: it cut off the attacker's ability to write to disk **two minutes and thirty-one seconds** after first contact, and neutralized the command-and-control dropper within seconds of activation. The attacker obtained a directory listing. That was the extent of it. The CMS core was untouched, the databases were clean, and the campaign was contained the same night.

Customer	Environment	Engaged	Outcome
Independent web agency Managed WordPress hosting Small team, enterprise-scale clients	~20 agents 5 client environments ~150 users Clustered Linux, shared network filesystem	Mid-incident, during an active compromise Agents live in ~24 hours	Attacker write access cut in 2 min 31 sec Databases and CMS core intact Campaign contained

THE CHALLENGE

A compromise that kept coming back

The agency ran WordPress at the application layer on clustered Linux web nodes sharing a distributed network filesystem. That shared filesystem is what made the intrusion so hard to shake: a file written to any one node appeared instantly on all of them, while each node's local temporary directory — untouched by any web-root cleanup — is exactly where the attackers staged their toolkits.

What the agency was actually facing

- **Initial access via stolen credentials, not a vulnerability.** A developer's CMS administrator credentials had been sold on a private market weeks earlier. Multiple attacker IPs across several countries simply logged in. There was no brute force to detect and no patch to apply.
- **Three weeks of dwell time** between the earliest confirmed artifact and full remediation.
- **Layered persistence.** Webshells and file managers in web roots; botnet toolkits staged outside every web root; a scheduled task that ran once and deleted itself; administrator accounts created across multiple sites within the same second and hidden from the user list; an API-created application password designed to survive password resets; and database-level persistence through an attacker-installed plugin.
- **Cleanups that would not hold.** A filesystem restore pulled from an already-infected source. Shells deleted in one hour were recreated within the next. Injected code in a theme file survived the first cleanup pass entirely.
- **Real exposure.** Dozens of hosted domains, a large population of administrator accounts, and form data belonging to the agency's clients' own customers.

WHY MONARX

The missing layer: runtime, on the server

THE FIRST MESSAGE TO MONARX SUPPORT

"I have an active threat against three servers and need immediate monitoring and remediation to ward off future attacks. We've done several rounds of hardening, but need to be sure the sites are 100% safe."

— The agency's founder

The agency had already layered on defenses: a CMS security plugin, a WAF, endpoint agents, uptime monitoring, and a CDN in front. None of it had stopped a determined operator working from valid credentials with shells already on disk. What was missing was something watching what code actually did at runtime, on the server, underneath the CMS.

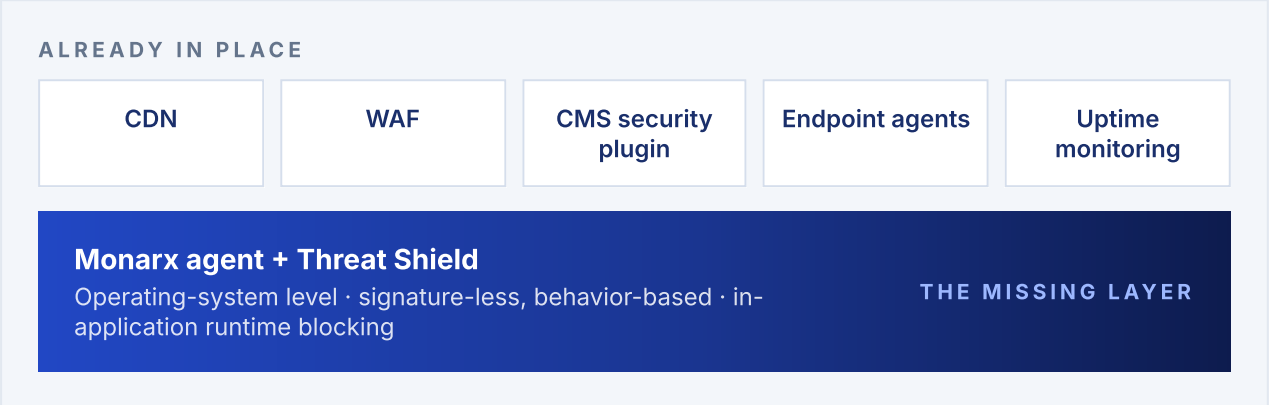


Figure 1. The agency's existing controls, and the runtime layer underneath the CMS that none of them covered.

That is the layer Monarx occupies. Its agent runs at the operating-system level with signature-less detection that evaluates behavior rather than file appearance, and its Threat Shield component provides in-application runtime protection — blocking malicious actions as they execute, whether or not that file has ever been seen before.

The part that mattered most: Monarx said yes

The agency's entire fleet would have landed at Monarx's published monthly minimum. There was no revenue case for dropping everything to help.

Monarx dropped everything anyway. The founder's first message arrived one evening; by the following afternoon an account executive had him on a call and enabled Active Protection and runtime blocking on the server already reporting. Within hours the agency had deployed agents across every node it was fighting for, and Monarx's on-call engineer and threat research team were engaged the same day. Inside about twenty-four hours, roughly twenty agents were online across five client environments with Active Protection and Threat Shield confirmed enabled on all of them.

~24h
from the founder's first message to a protected fleet

~20
agents online with Active Protection and Threat Shield enabled

5
client environments covered, before a contract existed

From first message to protected fleet

One evening	The founder's first message arrives at Monarx support.
Next afternoon	An account executive gets him on a call and enables Active Protection and runtime blocking on the server already reporting.
Same day	Monarx's on-call engineer and threat research team are engaged.
Within hours	Agents deployed across every node the agency was fighting for.
~24 hours	Roughly twenty agents online across five client environments, Active Protection and Threat Shield confirmed on all of them.

THE TEST

Just after midnight, the attackers came back

Days after deployment, the attackers came back. A webshell kit written to the shared filesystem earlier in the campaign — dropped by a scheduled task that then erased itself — was still sitting in a directory nobody had found. Just after midnight, an operator started using it.

Elapsed	Event	Control
T+0:00	Attacker confirms the shells are live, triggering a self-deleting dropper that pulls a large PHP remote-access trojan from an external command-and-control host.	—
T+0:00	The dropper deploys a botnet component to a second web node.	**Monarx neutralizes it within seconds**
T+0:34	Attacker executes commands through a password-protected file-manager shell and receives a directory listing of hosted sites.	—
T+2:31	**Monarx blocks the shell. The attacker can no longer write through it.**	**Blocked by Monarx**
T+3 to T+4	Attacker falls back to three secondary shells in the same directory. Every read succeeds; every write fails.	**Blocked by Monarx**
T+4:48	Attacker opens a file-manager interface and browses the hosted domain tree — read-only, with no ability to modify anything.	—
T+6	The agency spots the session in the access logs, captures forensic copies, and deletes the shell directory.	Manual
T+3h15	Staging toolkits cleared from every node. Active threat contained.	—

Table 1. Attacker session timeline, from the customer's contemporaneous incident report. Times are elapsed from first contact.

RESULTS

Read access only. Nothing written.

The distinction matters. The attacker had a working shell, valid filesystem permissions, and read access to dozens of domains. What he did not have, from T+2:31 onward, was the ability to write — no new backdoors, no re-seeded persistence, no modified pages, and no lateral spread to the other clusters through the shared filesystem. Every escalation path he reached for was already closed.

2 min 31 sec

from the attacker's first shell request to Monarx cutting off write access

The customer's own incident report, written that night: “Monarx blocked propagation within two minutes.”

Seconds

to neutralize the command-and-control dropper after it activated

7,800+

runtime events blocked by Threat Shield across the fleet in the first three days

What was protected

- **CMS core files:** no compromise.
- **Data exfiltration:** none beyond a directory listing of hosted site names.
- **Lateral spread** across the shared filesystem to the remaining nodes: none.
- **Databases:** clean.
- **File modification:** none outside the attacker's own shell directory.
- **Client-facing downtime** during the intrusion: none.

The agency retained every affected client relationship.

BEYOND THE INCIDENT

Everything around the block

The runtime blocking is the headline. What the customer talks about is everything around it.

A threat research team that worked the case, not the ticket

Monarx's threat research team reviewed flagged files directly and removed live injections found during that review. They identified injected redirect code and the exact file path it had been planted in — inside an archived site directory the agency had not prioritized. They advised on which exploits to patch against, walked through the three CMS access paths that needed protecting, and recommended concrete controls: IP allowlisting, rate limiting, zero-trust access, and a CAPTCHA challenge in front of the login. They also offered an open channel: send us anything we do not detect, and we will build a signature for it.

TWO DAYS AFTER DEPLOYMENT

“So far, it seems things have been keeping the attackers at bay. Wondering what else I can do.”

— The agency's founder

Availability that matched the urgency

Weekend syncs. An on-call engineer looped in the same day. A standing invitation to reach the team by chat over a weekend when the customer was monitoring a fleet he no longer fully trusted. Small things — which is exactly what a very small team in the middle of an incident notices.

Transparency, tested

Two months later, Monarx's CTO emailed the customer directly to disclose a critical privilege-escalation vulnerability in an older version of one of Monarx's own components — confirming which versions were affected, that no attack had leveraged it, and exactly what to do about it. The same message disclosed a separate race condition the team had found in its own agent during that audit and hardened, along with the internal process changes made to prevent a recurrence.

A vendor volunteering a severe flaw in its own product, unprompted, to a customer at the bottom of its price band is not a common experience — and for a security partner it may be the most meaningful proof point of all.

IN THE FOUNDER'S WORDS

“Monarx had no commercial reason to help us at the scale we were at. They helped anyway — same day, before there was a contract. Days later their agent cut a live attacker off in under three minutes. I don't think that's a coincidence, and I don't think we'd have had the same outcome without them.”

— The agency's founder

WHAT GENERALIZES

Three lessons for any hosting fleet

1 **Runtime beats reputation when the shells are already there.**

The webshells had been written to disk days before the attack and had survived two separate malware scans — one of which flagged several files as potential webshells and dismissed all of them as false positives. Signature and file-scanning approaches did not find them. What stopped the attack was watching what the code did the moment it ran.

2 **Server-level placement covers what application plugins cannot.**

The attacker's staging directories sat outside every web root, on a filesystem no CMS plugin can see. The agent operates below that line.

3 **Speed of deployment is itself a security control.**

Monarx went from first contact to a fully protected fleet in roughly a day. Had that taken a normal procurement cycle, the agent would not have been running when the attacker came back.

NEXT STEP

Runtime protection for your hosting fleet

monarx

monarx.com/get-started →