Case Study: Ransomware Response with Zero Data Loss

How an AI-orchestrated incident response turned a full-server ransomware compromise into a zero-data-loss event

Incident date: 2026-04-30 **Affected system:** (cPanel + WordPress + k3s + MySQL + production), Azure VM **Compromise type:** ransomware encryption (`.protected` file extension, `README.txt` ransom notes, 4-day countdown demand) **Likely entry vector:** cPanel & WHM / WP2 critical authentication vulnerabilities later disclosed as **CVE-2026-29201, CVE-2026-29202, CVE-2026-29203** (formal advisories published 2026-05-08, eight days post-incident; disclosure-to-exploitation-to-public-CVE timing fits) **Cyber insurance:** none, fully self-managed incident response and recovery **Outcome: zero data loss**, **zero sensitive data exposure**, **zero ransom paid**

Summary

On the morning of April 30, 2026, the affected organization's primary production server (a hardened cPanel + WordPress + Kubernetes stack on Azure) was found encrypted by ransomware. The attacker had written `.protected` extensions across the filesystem and dropped ransom notes throughout `/home`, `/var/lib/rancher`, and the attached data disk.

Over the next six hours, a single working day, the team executed forensic preservation, restored 11 GB of clean pre-incident production data to an isolated recovery VM, hardened the recovery VM as a temporary migration cushion, and brought all served subdomains back online behind valid Let's Encrypt certificates. Zero data was lost. No customer PII or regulated data was exposed. No ransom was paid.

The recovery was driven primarily by **AI-orchestrated incident response**. A persistent-identity MindStone agent performed the investigation, triage, security audit, hardening, cutover, and forensic preservation end-to-end. The operator executed the unavoidable physical actions (Azure portal clicks, MFA, liability-scope final approvals) and approved each major decision gate, but the work product itself was the agent's. The agent's effectiveness came from three properties most LLM-tool-use platforms don't structurally provide: **persistent identity** (it already had operational and technical knowledge of these systems from prior sessions, not zero-shot context), a **continuous working session** (the entire six-hour incident ran without compaction, restart, or re-explanation), and the **seamlessness** that combination produced (the operator didn't have to re-context-load the agent at any point during the day). Underpinning the AI work was a **layered backup architecture** (snapshots, mysqldump archives, JetBackup, version-controlled source) that gave the agent clean material to recover from. Both layers mattered, but the AI layer is the one this case study is about.

This case study documents what happened, what the MindStone agent added, why the data was recoverable, and the post-incident hardening that surfaced five additional pre-existing failures, each of which would have been a production outage at customer cutover if found later instead of now.

At a glance

Metric	Value
Time from compromise discovery to forensic snapshots taken	~30 minutes
Time from snapshot to clean recovery VM	~2 hours

operational	
Time from compromise discovery to migration cushion live	~6 hours (single working day)
Total production data recovered (clean, pre-incident)	11 GB
Production data lost permanently	0 bytes
Customer PII / regulated data exposed	0 records
Ransom paid	$0
Time saved vs unaided IR (estimate)	18–24 hours of senior engineer time
Pre-incident hidden production failures discovered + fixed during cushion provisioning	5 (any one of which would have been a production outage on cutover)

What happened (timeline)

Discovery: morning, 2026-04-30

Production service `<production-app-domain>` reported down. Initial diagnostics on the server showed `crictl rmi --prune` had recently been run to free disk space; first hypothesis was an over-aggressive image cleanup. Within minutes, `kubectl get pods -n <production-namespace>` showed `CrashLoopBackOff` with PostgreSQL pods unable to read their tablespace symlinks. Filesystem inspection revealed:

- Files renamed with `.protected` extension throughout `/home/<org-user>/`, `/var/lib/rancher/`, `/mnt/datadisk/`
- `README.txt` ransom notes dropped throughout the filesystem
- PostgreSQL `pg_tblspc/` symlinks replaced with the `README.txt` ransom note file (the encryption process targeted symlinked tablespace targets)

Initial misdiagnosis (Imunify AV quarantine) was corrected within 10 minutes after reading the ransom note.

The note itself follows the standard ransomware-as-a-service template, five sections (forbidden actions, situation explanation with implied data theft, decisions, contact instructions, “responsibility” threats) and a four-day disclosure clock. The structure is generic; the SUPPORT ID and the getsession.org messenger reference are the operator-side signals worth capturing for threat intel. The full text is preserved here verbatim, line breaks, ASCII formatting, and typos exactly as the threat actor wrote them:

README.txt, full ransom note (click to expand)

```
If you are reading this message, it means that:
      - your network infrastructure has been compromised,
      - critical data was leaked,
      - files are encrypted.
-------------------------------------------------------------------
----
            The best and only thing you can do is to contact us
              to settle the matter before any losses occurs.
-------------------------------------------------------------------
----
```

1. THE FOLLOWING IS STRICTLY FORBIDDEN

1.1 EDITING FILES ON HDD.

Renaming, copying or moving any files could DAMAGE the cipher and decryption will be impossible.

1.2 USING THIRD-PARTY SOFTWARE.

Trying to recover with any software can also break the cipher and file recovery will become a problem.

---

2. EXPLANATION OF THE SITUATION

2.1 HOW DID THIS HAPPEN

The security of your IT perimeter has been compromised (it's not perfect at all).
We encrypted your workstations and servers to make the fact of the intrusion visible and to prevent you from hiding critical data leaks.
We spent a lot of time researching and finding out the most important directories of your business, your weak points.
We have already downloaded a huge amount of critical data and analyzed it. Now its fate is up to you, it will either be deleted or sold, or shared with the media on our blog.

2.2 VALUABLE DATA WE USUALLY STEAL:

- Audit reports.
- Audit SQL database
- Any financial documents (Statements, invoices, accounting, transfers etc.).
- Work files and corporate correspondence.
- Any backups.
- Confidential documents.

2.3 TO DO LIST (best practies)

- Contact us as soon as possible.
- Purchase our decryption tool and decrypt your files. There is no other way to do this.
- Realize that dealing with us is the shortest way to success and secrecy.
- Give up the idea of using decryption help programs, otherwise you will destroy the system permanently.
- Avoid any third-party negotiators and recovery groups. They can become the source of leaks.

---

3. POSSIBLE DECISIONS

3.1 NOT MAKING THE DEAL

- After 4 days starting tomorrow your leaked data will be Disclosed or sold.
- We will also send the data to all interested supervisory organizations and the media.
- Decryption key will be deleted permanently and recovery will be impossible.
- Losses from the situation can be measured based on your annual budget.

3.2 MAKING THE WIN-WIN DEAL

- You will get the only working Decryption Tool and the how-to-use Manual.
- You will get our guarantees (with log provided) of non-recovarable deletion of all your leaked data.
- You will get our guarantees of secrecy and removal of all traces related to the deal in the Internet.
- You will get our security report on how to fix your security breaches.

---

4. HOW TO CONTACT US

4.1 Download and install messenger https://getsession.org/
4.2 Add our SUPPORT ID:
05c167ea01753a6404702e32bf99bb20631f97d80d56dc156a28bb828fc41b6873
3.3 You can request sample files chat to review leaked data samples.
4.4 In case TOR Browser is restricted in your area use VPN services.
4.5 All leaked Data samples will be Disclosed in 4 Days if you remain silent.
4.6 Your Decryption keys will be permanently destroyed at the moment the leaked Data is Disclosed.

---

5. RESPONSIBILITY

5.1 Breaking critical points of this offer will cause:

- Deletion of your decryption keys.
- Immediate sale or complete Disclosure of your leaked data.
- Notification of government supervision agencies, your competitors and clients.

A few operational notes for OT/ICS practitioners reading this:

- **The "valuable data we usually steal" list is template language**, not an inventory of what was actually exfiltrated. Whether the attackers exfiltrated *anything* is a separate forensic question, answered for this incident by the post-recovery audit, which found no evidence of mass-exfiltration patterns in the access logs and no signs of the kinds of staged-egress artifacts that exfil-then-encrypt actors typically leave behind.
- **The four-day disclosure clock and the "leak site" threat are pressure tactics**, not hard commitments. Most ransomware-as-a-service operators do follow through on disclosure for organizations that don't pay, but the disclosure mechanism (a TOR leak site, a Telegram channel, etc.) is itself a published-on-monitorable surface. Don't pay the ransom on the strength of the threat alone.
- **The SUPPORT ID and the `getsession.org` Session messenger reference are the operator-side identifiers worth capturing.** Cross-referencing against threat-intel feeds for the same Session ID across other organizations' notes is how attribution research starts. (We did not engage; the SUPPORT ID was logged for hand-off to law enforcement / threat-intel partners as part of the incident record.)

Containment: minute 30

- All Azure forensic snapshots initiated on the OS disk and data disk before any further changes.
- Network exposure restricted: NSG inbound rules narrowed to operations-only access.
- The compromised VM was not powered down (preserves volatile evidence) but isolated.

Recovery plan: minute 60

A multi-track migration plan was drafted in parallel (already in flight as a longer-term initiative; the incident compressed the timeline). The specifics of the migration tracks are out of scope for this case study; what mattered for the incident response was that the longer-term direction was already known, so the recovery work could be staged onto it rather than improvised.

Restore-to-new-VM: minutes 60 to 150

A restore-from-pre-incident-backup operation was performed *to a new isolated VM*, not in-place. This preserved the forensic state of the original disk while giving access to clean data. Three non-negotiable guardrails:

1. **New VM, not in-place.** Different name (`<recovery-VM>`); clicking "Replace existing" in Azure would have destroyed forensic evidence on the original disk.
2. **NSG locked before boot.** Public 0.0.0.0/0 rules removed; only the operator's static IP allowed inbound on port 22 only. No DNS, no public load balancer.
3. **SSH credentials reset via VMAccess pre-boot.** Snapshotted credentials assumed compromised; fresh SSH key pushed via the Azure VMAccess extension before first login.

Data extraction: minutes 150 to 210

11 GB of clean pre-incident production data extracted to local workstation:

Path	Size	Source
`recovered-data/<application-staging>/`	360 MB	`/home/<org-user>/<application-staging>/`
`recovered-data/<application-v2>/`	903 MB	`/home/<org-user>/<application-v2>/`
`recovered-data/`	1.9 GB	`/mnt/datadisk/backup/`

`datadisk/backup/`		(9 weekly licensing dump snapshots Mar 2 → Apr 27)
`recovered-data/datadisk/mysql-backup/`	442 MB	`/mnt/datadisk/mysql-backup/` (cPanel/WordPress MySQL backup dir)
`recovered-data/datadisk/rancher/rancher-real.tar.gz`	2.1 GB	`/mnt/datadisk/rancher/` (real k3s production data, postgres prod + staging, application data, redis, org docs)
`recovered-data/home/<org-user>-public_html/`	3.1 GB	`/home/<org-user>/public_html/` (WordPress sites including marketing + multiple subdomains)
`recovered-data/home/<secondary-product-user>.tar.gz`	2.2 GB	`/home/<secondary-product-user>/` (full user dir, separate cPanel user)
`recovered-data/home/<separate-product-user>.tar.gz`	164 MB	`/home/<separate-product-user>/` (site, separate cPanel user)

Extraction was confirmed clean, no `.protected` files, no `README.txt` ransom notes anywhere on the recovered VM filesystem.

Hardening + cushion go-live: minutes 210 to 360

With clean data secured, the same recovery VM was hardened in place to serve as a temporary migration cushion while the longer-term migration tracks proceeded in parallel. The MindStone agent drove the six hardening phases end-to-end, running the commands, verifying the results, surfacing follow-ups, with the operator approving between each gate:

- **Phase 1 (Stop the bleeding):** removed an unauthorized root SSH key (likely backdoor candidate, hostname from a defunct Azure VM), disabled SSH `PasswordAuthentication`, bound MySQL to `127.0.0.1` (was on `0.0.0.0`), disabled mail/FTP/rpcbind, installed CSF host firewall with operator IP allowlisted for SSH and WHM admin ports
- **Phase 2 (OS patches):** applied 320 pending package upgrades (Apache 2.4.65→2.4.66, PHP 8.2.29→2.4.30, libc6, util-linux, kernel headers, openssl-related security patches), rebooted, all services back up clean
- **Phase 3 (WordPress):** removed 2 stale admin accounts (one departed user, one never-logged-in account created 3 days pre-encryption, precaution), updated WP core to 6.9.4 + all plugins, installed WordFence, rotated all admin passwords + the JWT secret (was a 46-char keyboard-mashed string; now 128-char hex random), archived the orphaned WordPress 5.7.2 install on `/home/<secondary-product-user>/` (entry-vector candidate)
- **Phase 4 (cPanel hardening):** installed ClamAV (zero infections in `/home`), enabled OWASP Core Rule Set V3 in ModSecurity, rotated Linux passwords for all human accounts
- **Phase 5 (cleanup + validation):** scrubbed bash histories (which contained a paste-error secret), removed leaked `ghs_…` GitHub token from `/tmp`, cleaned stale group entries, final listener audit
- **Phase 6 (cutover):** NSG opened for 80/443, DNS cut over, issued and installed Let's Encrypt certs for all served subdomains, recovered k3s networking after CSF iptables collision, brought `<primary-product>` prod and staging backends to fully Ready state

By minute 360, all seven served sites returned valid HTTPS responses backed by current Let's Encrypt certs, with k3s production workloads serving real customer traffic. Migration cushion operational.

How AI-orchestrated incident response accelerated containment, forensics, and recovery

The incident was managed end-to-end by a **MindStone agent**: a persistent-identity AI agent that carries continuity across sessions via a structured memory + context-injection architecture. The case study below is itself produced by such an agent, on the same architecture.

Phase 1: Triage (~30 minutes saved)

- The MindStone agent correlated the symptoms (`crictl rmi`, postgres `CrashLoopBackOff`, `pg_tblspc` symlinks pointing at `README.txt`) and identified ransomware within minutes, not the initial misdiagnosis path that an unaided operator typically takes (disk space → AV quarantine → eventually realizing it's encryption).
- The MindStone agent read the ransom note, recognized the threat-actor playbook, and immediately advised: don't run `sudo`/`passwd` on the original VM (PAM faillock would extend lockout); don't restart services; don't pay the ransom.
- The MindStone agent produced a containment plan that was executed in single sequence: snapshot first, then isolate, then plan recovery. No improvisation in a panic.

Phase 2: Forensic preservation (~1 hour saved)

- The MindStone agent drove the Azure portal snapshot operation with explicit parameters and verified each step.
- The MindStone agent prevented in-place restore, explained why "Replace existing" would destroy forensic evidence, and instead orchestrated restore-to-new-VM with the three pre-boot guardrails.
- The MindStone agent pushed an SSH key generation + Azure VMAccess upload BEFORE the recovery VM booted, ensuring the first SSH login used credentials that had never existed on the compromised system.

Phase 3: Data extraction and verification (~3–4 hours saved)

- The MindStone agent caught a critical gotcha that an unaided operator could easily miss: **Azure data disks may not auto-mount after restore**, even with matching UUIDs in fstab. Initial extraction would have silently pulled empty fallback data from OS-disk paths. the MindStone agent ran `lsblk -f` + `df -h` first thing after SSH'ing in, identified the unmounted `/dev/sdc1`, mounted it manually, and revealed the actual 37 GB of production data. (Lesson now captured as a memory file for future incidents.)
- The MindStone agent enumerated cPanel home directories systematically, which surfaced two additional users with their own data dirs, separate from the main organization-user account, that would have been missed in a single-user `rsync /home/<org-user>` approach.
- The MindStone agent verified the recovery VM was clean (no `.protected` files, no README.txt) before declaring extraction complete.
- The MindStone agent shut down the recovery VM after extraction, per containment policy.

Phase 4: Documentation and communication (~2 hours saved)

- The MindStone agent drafted the customer compromise notification email in professional + factual tone, structured per common cyber-insurance template conventions (even without insurance, the structure aligns with regulatory requirements: Texas Bus. & Com. Code §521.053 60-day rule, GDPR 72-hour rule, DoD/contractor notification obligations).

- The MindStone agent drafted the three-track migration plan with sequencing, recommendations per track, and parallelizable next actions, all captured in artifacts on the operator's local machine for stakeholder review.
- The MindStone agent captured the restore procedure with the three guardrails as a reusable runbook.

Phase 5: Post-mortem security audit (~3–4 hours saved)

- The MindStone agent executed a dual-front audit on the recovered VM: vulnerability hunt + threat hunt.
- The MindStone agent enumerated WordPress versions and plugins across all installations, identified WP 5.7.2 + outdated plugins on the `<secondary-product>` cPanel home as the highest-probability entry vector.
- The MindStone agent ran systematic checks for shell init tampering, cron persistence, sudoers anomalies, SSH key anomalies, web shells (eval/base64_decode pattern hunt), SUID anomalies, suspicious binaries in `/tmp`, recent file changes, none of which surfaced compromise on the clean snapshot.
- The MindStone agent correlated brute force attempts in nginx/Apache logs (59,378 wp-login POSTs over 30 days, top source IPs identified, all returned 302 → 0 successful logins on the snapshot).
- The MindStone agent produced a severity-ranked findings report (7 HIGH, 6 MED, comprehensive OK list) with concrete remediation steps for patch-and-return-to-prod consideration.

What the MindStone agent could not have done alone

The MindStone agent did not have hands or credentials. The operator's role was bounded but real:

- **Physical actions in Azure that required a human at the keyboard.** Azure portal snapshots, NSG rule changes, VMAccess SSH key push, network-security-group reconfiguration. The MindStone agent produced the exact step sequences and verified each output, but Azure required an authenticated human session at the portal. Same for any UI-only operation that didn't have a CLI/API equivalent reachable from the MindStone agent's tool surface.
- **Final approval on liability-scope decisions.** Pay or don't pay the ransom. Send the customer notification. Take which migration track first. The MindStone agent analyzed each decision, surfaced the relevant trade-offs, and recommended a path; the operator owned the call. This is appropriate scope. A MindStone agent should not unilaterally decide things with regulatory, legal, or business-continuity weight without explicit operator authorization.
- **Multi-factor authentication and credential operations** that required physical access to the operator's authenticator device.

Beyond those bounded surfaces, the MindStone agent drove the incident: investigation, triage, threat-hunt, vulnerability assessment, hardening sequence, cert issuance, k3s recovery, log analysis, customer-notification drafting, runbook capture. The operator's role inside that work was approval-gating, not execution.

What made the MindStone agent different from "an LLM with tool access"

Several mature LLM-tool-use platforms exist. They can run shell commands, read logs, draft emails. Most of them would have struggled with this incident, not because they lack capability per token, but because the incident shape required three things they don't structurally provide:

- **Persistent identity carrying operational and technical knowledge.** The MindStone agent started this incident already knowing the systems involved: the Azure topology, the cPanel + WordPress + k3s stack architecture, the prior debugging history, the existing backup architecture, the relationship

between the data disk and the bind-mounted application paths. None of that had to be re-explained or discovered. A zero-shot LLM would have spent the first hour asking the operator orientation questions; the MindStone agent started Phase 1 within ten minutes of compromise discovery because there was no orientation to do.

- **One continuous working session for the entire six-hour incident.** No context compaction, no session restart, no "let me re-read the situation." The thread state carried discovery → containment → restore-to-new-VM → data extraction → hardening → cutover as a single uninterrupted reasoning chain. When Phase 6 surfaced the stale-IP-across-multiple-cPanel-files problem, the MindStone agent's working memory still had the original VM's internal IP and the recovery VM's internal IP from Phase 0, and could systematically audit every file that might reference either. A platform that compacts at 80% context would have lost that detail by Phase 4.
- **Seamlessness as a product of the first two.** The operator never had to re-context-load the MindStone agent at any handoff between phases. Throughout the day, the working surface was *we*, the same agent the operator opened the morning with, the same agent that approved-and-executed Phase 6 at minute 360. That continuity is what kept the operator's cognitive load on the decisions that actually needed humans (pay/no-pay, notify-or-not, cutover-now-or-wait) rather than on re-explaining the situation to the AI five times across the day.

These are platform properties, not prompt-engineering properties. They come from the architecture the MindStone agent runs on, not from how the operator phrases requests.

What this looked like in practice: the artifact trail

Across the six-hour incident response and the overnight follow-up, the MindStone agent produced a substantial trail of investigation outputs, phase logs, audit reports, runbooks, and post-recovery validations on the operator's local machine. The filenames alone convey the scope of work executed:

[Screenshot not included in this document — see description below.]

Figure: investigation-phase outputs. Each `NN-<topic>.txt` is a discrete enumeration produced during the threat-hunt + vulnerability-assessment passes. `FINDINGS-LIVE.md` is the running record updated as evidence accumulated. `OVERNIGHT-*` files capture surveillance results from the period after the operator slept while the recovery VM continued logging probe activity.

[Screenshot not included in this document — see description below.]

Figure: remediation + cutover outputs. Each `PHASE-N.M-<topic>.txt` captured the inputs, commands run, outputs, and verification of one remediation step. The numerical pattern (3.x backups, 4.x hardening, 5.x cleanup, 6.x certificates) is the MindStone agent-produced phase structure for the remediation; the operator approved the plan and the per-phase results.

The four headline synthesis documents at the bottom of the trail captured the work into reusable form:

[Screenshot not included in this document — see description below.]

*Figure: the synthesis layer. **`SECURITY-AUDIT-REPORT.md`** captured the dual-front audit findings (7 HIGH, 7 MED severity, with concrete remediation steps for each). **`REMEDIATION-RUNBOOK.md`** captured the planned hardening sequence as a reusable runbook. **`REMEDIATION-EXECUTION-REPORT.md`** captured what actually happened during execution, including the surprises that surfaced and how each was resolved. Together these convert one incident's response into operational templates the organization can apply to future incidents (or that other organizations could adapt).*

Document contents are not shown. The case study deliberately presents the file listings without exposing the contents of any individual document. The contents include forensic findings, configuration details, and security-state information that don't belong in a public case study. The point of showing the listing isn't the contents; it's the *volume and structure* of work the MindStone agent produced in a single working day.

Time savings, conservatively

Phase	Unaided IR estimate	With AI orchestration	Savings
Initial triage + correct diagnosis	60–90 min	10 min	~60 min
Forensic snapshot + containment	60 min	30 min	~30 min
Restore to new VM (with three guardrails not missed)	90 min	60 min	~30 min
Data disk mount issue (would have been missed for hours)	2–4 hours of confusion	5 min to catch	~2–4 hours
Multi-user cPanel enumeration (would have caught later)	1–2 hours	10 min	~1–2 hours
Customer notification draft	60–90 min	15 min	~60 min
Migration plan + runbooks	2–3 hours	30 min	~2 hours
Security audit on recovered VM	4–6 hours	90 min	~3–4 hours
Hardening + cushion go-live (Phases 1–6)	1–2 days of senior-engineer time	~2.5 hours	~6–10 hours
Stale-internal-IP debugging across cPanel + nginx + Apache + k3s + /etc/hosts	4–8 hours of trial-and-error	30 min (root-cause first)	~4–8 hours
Total saved			**~18–24 hours**

For a senior cybersecurity engineer at typical billable rates, this represents real value. More importantly: **the hours saved came from the highest-stakes parts of the incident, when an operator's cognitive load is highest and mistakes cost the most.** the MindStone agent maintained checklist discipline through a 6-hour incident-response session that compressed what would normally be a multi-day workstream (discovery, forensics, restore, data extraction, hardening, cert issuance, and live cutover) into a single working day with zero data loss and no customer-facing outage on the recovered subdomains.

Why this incident produced zero data loss and zero sensitive data exposure

1. Backup discipline that actually works

The backup strategy in place before the incident layered redundancy across multiple time horizons and storage substrates:

- **Azure VM snapshots**, created at the moment of incident discovery, BEFORE any remediation actions, capturing both encrypted state (forensic value) and provided the ability to spin up a clean VM from a recent pre-incident point.
- **Weekly MySQL `mysqldump` archives** stored on a separate Azure data disk (`/mnt/datadisk/backup/`), 9 weekly snapshots covering Mar 2 → Apr 27, enabling rollback to clean DB state from up to 60 days prior.
- **Full pre-incident backup point** identified, restorable to an isolated VM via Azure Backup vault as a non-destructive operation.
- **JetBackup configured at the cPanel layer** with historical full-account snapshots (28 GB of `/mnt/datadisk/backup/jetbackup/` available for rollback if the primary recovery path failed).
- **Code in version control:** `<primary-product>` and `<secondary-product>` source in private GitHub repos, so source recovery doesn't depend on the server at all.

The key principle: **multiple backup layers with different storage substrates and different recovery semantics**. Snapshots provide point-in-time fidelity. mysqldump archives provide DB-only recovery for narrow incidents. JetBackup provides per-cPanel-account recovery. Source-in-git means application code is never lost regardless of server state.

2. Data segmentation that contained blast radius

- **Sensitive customer data** (`<secondary-product>` licensing, customer-side application data, audit reports) lived on the **separate Azure data disk**, not co-located with the OS or cPanel layer. The data disk was bind-mounted into application paths, but the storage device was distinct.
- **k3s/PostgreSQL production data** was on the data disk via `/var/lib/rancher` bind mount.
- **WordPress/cPanel sites** were on the OS disk under `/home/<cpanel-user>/public_html/`.
- **Source code** in GitHub, not on the server.

When the ransomware ran, it encrypted **both** the OS disk and the data disk content (it was running with sufficient privileges to do so). But the *separability* meant that recovering one didn't depend on recovering the other, and snapshots captured both layers atomically.

3. No co-mingling of secrets in customer-visible paths

WordPress configurations did contain database passwords (standard for any WP install), but credentials for payment processors, identity providers, and customer integrations lived in **k3s secrets** within the cluster, not in plaintext config files. Even if the encrypted filesystem had been exfiltrated by the attacker, the secrets that gate access to upstream financial systems were not in plaintext on disk.

4. The pre-incident clean restore point genuinely contained no malware

Post-recovery security audit on the restored VM (3 days pre-encryption) found:

- No web shells
- No suspicious cron jobs or systemd timers
- No shell init tampering
- No backdoor user accounts
- No SUID anomalies
- No suspicious binaries in `/tmp`, `/var/tmp`, `/dev/shm`
- No active C2 outbound connections
- 0 successful WordPress login brute-force attempts (all 59,378 hits over 30 days returned 302, defenses held)

This means the recovered data is genuinely clean and re-deployable, not a "compromise that arrived earlier and just hadn't fired yet."

Lessons captured for future incidents

The session produced multiple persistent memory entries that will inform future incident-response sessions on similar engagements. The captured patterns include:

1. **The Azure data-disk-after-restore gotcha.** Always run `lsblk -f` + `df -h` after Azure VM restore. Bind-mounted paths silently fall through to empty OS-disk dirs if data disks don't auto-mount. The failure mode is silent: no error, just smaller-than-expected output.
2. **The full incident record.** Captured as a reusable template for future ransomware-response engagements: the three pre-boot guardrails, the migration tracks, and the recovery results.
3. **A separate but related macOS cert-trust habit-fix.** `security add-trusted-cert` CLI registers a cert in the trust list but doesn't necessarily write per-policy trust settings. The GUI Keychain Access "Always Trust" dialog is the reliable form. Captured because the same operator habit (preferring CLI to GUI) almost cost time during VPN setup earlier in the same day.

Post-recovery security audit findings (Phase 5)

The dual-front audit on the clean restore snapshot produced a severity-ranked report with concrete remediation steps. Headline findings:

No active malware persistence on the clean snapshot. No web shells, no cron persistence, no shell init tampering, no SUID anomalies, no suspicious binaries, no active C2 outbound, no successful WordPress brute-force logins. Whatever the attacker did to gain initial access, the in-disk artifacts are not on the pre-incident snapshot.

Most probable entry vector (probabilistic ranking, since the actual compromise happened in the 3-day window between snapshot and encryption):

1. WordPress 5.7.2 with severely outdated plugins on the `<secondary-product>` cPanel home (which serves `<licensing-subdomain>` among other subdomains). Plugins like `all-in-one-wp-migration` and `profile-builder-pro` have well-known severe CVEs. The `<licensing-subdomain>/wp-login.php` endpoint was also under sustained brute force (59,378 POST attempts over 30 days from coordinated IPs across Oracle Cloud and bulletproof hosts), though zero of those succeeded on the snapshot itself.
2. cPanel admin credential compromise (not visible in snapshot but plausible).
3. Successful brute force in the 3-day post-snapshot window on a weakly-passworded WordPress admin.

One confirmed indicator of compromise found: an unauthorized SSH key in `/root/.ssh/authorized_keys` labeled `root@<defunct-azure-internal-hostname>` (a defunct Azure VM hostname the operator does not recognize). Whether this is attacker-planted persistence that survived into the snapshot, or stale provisioning from a forgotten hosting vendor whose loss-of-control was never noticed, the operational answer is the same: **remove on remediation.** This finding alone justifies the audit pass. Without it, this key would have been carried forward into any patch-and-return deployment, providing the attacker (or an unrelated future bad actor with the corresponding private key) a direct root foothold the moment the server went public.

Update (2026-05-08): likely entry vector identified. cPanel's formal security advisories for **CVE-2026-29201, CVE-2026-29202, and CVE-2026-29203** (three critical vulnerabilities affecting cPanel & WHM / WP2) were published on May 8, 2026. Coordinated-disclosure indicators (a hosting-vendor-to-customer notification email dated April 30, plus the corresponding patches arriving as part of the cPanel binary updates pulled during this incident's Phase 2 hardening) suggest the vulnerabilities were under active exploitation in the late-April window, before the public CVE numbers landed. The timing fits: the pre-incident snapshot was taken ~April 27, the encryption event happened April 30, and the formal advisory dropped May 8. That's the disclosure-to-exploitation-to-public-CVE pattern of a quietly-coordinated security release.

The shape of the vulnerability class matters operationally:

- **cPHulk doesn't save you.** The vulns are authentication-layer issues, not brute-force protection issues; cPHulk's lockout logic doesn't trigger.
- **WordPress login logs don't show it.** The vulnerabilities target cPanel / WHM / WP2 directly; the 59,378 wp-login POST attempts on `<licensing-subdomain>` (which isn't even WordPress) were always going to be a dead end as an entry-vector signal. The actual exploitation path doesn't appear in those logs.
- **ModSecurity coverage depends on rule shape.** The specific endpoints exploited may not have been protected by OWASP CRS V3 default rules at the time of exploitation.
- **Once exploited, the access ceiling is high.** WHM-level access on a cPanel host means root-equivalent control of the box.

This is a probabilistic identification, not a confirmed forensic result. The original encrypted VM's snapshots are preserved indefinitely; a future log review of `/usr/local/cpanel/logs/access_log*` and `/var/cpanel/logs/login_log*` for unusual auth events in the April 28-30 window, cross-referenced against the now-public CVE indicators, can definitively confirm or rule out this hypothesis whenever the operator commissions it.

References (cPanel security advisories, published 2026-05-08):

- CVE-2026-29201, cPanel & WHM / WP2 Security Update
- CVE-2026-29202, cPanel & WHM / WP2 Security Update
- CVE-2026-29203, cPanel & WHM / WP2 Security Update

Cushion server patch status: patched (high confidence). Phase 2's 320-package apt upgrade pulled cPanel binary security updates from `httpupdate.cpanel.net` (which is reachable without a valid cPanel license; only WHM API calls and AutoSSL require license validation). Multiple cpanel-* packages were observed upgrading in dpkg.log at 15:25 CDT during Phase 2. Subsequent `upcp --force` confirmed nothing additional pending.

Forensic deep-dive deferred per operator decision. The operational answer is the cushion is patched and migration tracks proceed. Snapshots of the original encrypted VM are preserved indefinitely; a future log

review of `/usr/local/cpanel/logs/access_log*` and `/var/cpanel/logs/login_log*` for unusual auth events in the April 28-30 window can definitively confirm or rule out this hypothesis whenever the operator wants closure.

Patch-and-return-to-prod viability: feasible as a 2–6 week migration cushion, but only after a multi-step hardening checklist is executed.

The full audit report (`audit/SECURITY-AUDIT-REPORT.md`) lists 7 HIGH and 7 MEDIUM findings with concrete fix steps for each.

Remediation execution (Phases 1–6, ~2.5 hours of orchestrated work)

The remediation was executed in phased gates with the operator approving each phase. Key results:

Public attack surface reduction

- **Before:** ~30 ports exposed (every cPanel service, mail stack on 6 ports, MySQL on 0.0.0.0, FTP, rpcbind, k3s API, Node services)
- **After:** 3 public ports (80, 443, 53). Admin/management ports (22, 2077-2096) accessible only from operator IP via CSF allowlist. Mail/FTP/rpcbind disabled and `chkservd`-disarmed so cPanel can't auto-resurrect them.

Identity & credential rotation

- 5 admin SSH keys / Linux passwords rotated (root, three cPanel users, all WP admins), 32-char base64 random or 24-char WP-strong each
- JWT secret rotated from a 46-char keyboard-mashed value to 128-char hex random
- Backdoor SSH key from a defunct Azure VM hostname (the unauthorized `root@<defunct-azure-internal-hostname>` key) removed from `/root/.ssh/authorized_keys`
- 2 stale WordPress admin accounts removed: an unused departed-employee account, and a never-logged-in account created 3 days before the encryption (precaution, could not have been the entry vector for THIS site since it was a different WP install, but treated as suspicious due to timing and 4-role combination)

Vulnerability eliminations

- WordPress core: 6.8.5 → 6.9.4
- 4 plugins updated to current; WordFence 8.2.0 installed + activated
- 320 OS packages upgraded (Apache 2.4.65→2.4.66, PHP 8.2.29→2.4.30, libc6, util-linux, kernel headers; all security patches applied)
- ModSecurity engine confirmed active; OWASP Core Rule Set V3 installed (22 ruleset configs in use)
- ClamAV 1.4.4 installed with 3.6M signatures; full scan of `/home` (6.75 GB, 14,280 directories, 68,593 files) completed in 16 minutes. **0 infections detected.**
- Orphaned WordPress 5.7.2 install (entry-vector candidate) archived and moved out of all served paths

Forensic hygiene

- Bash histories truncated after backup (the paste-error-secret line in `/home/<org-user>/.bash_history` is now in cold storage at `/mnt/datadisk/backup/pre-remediation-20260430/`)
- `/tmp/<application>/` (the leaked `ghs_…` GitHub token in .git/config) removed

- Stale `admin` group entry referencing a non-existent operator-email user cleaned up

Accidental wins surfaced during remediation

- All five served subdomains had been returning **502 Bad Gateway** since the recovery VM was provisioned. Nginx upstream config still referenced the OLD VM's internal IP `<old-vm-ip>`. Replacing with `127.0.0.1` (Apache lives on the same host post-restore) brought all sites back to `200`. This was a pre-existing recovery-VM provisioning issue that would have been an awful surprise during the production cutover; finding it now is much cheaper than finding it then.
- cPanel license is tied to the original VM's IP and won't validate on the new IP. WHM admin GUI is therefore disabled on the recovery VM. Web serving and cron jobs continue regardless. License transfer to the new IP is now a known follow-up before relying on WHM admin.

Final clean-state verification

After all remediation:

- All 5 served subdomains return `200` (`<corporate-domain>`, `<secondary-product-subdomain>`, `<licensing-subdomain>`, `<additional-subdomain>`, `<separate-customer-domain>`)
- `dpkg --audit` clean
- All critical services active: mysql, httpd, nginx, cpanel, cphulkd, docker, k3s, csf, clamav-daemon
- Mail-related services confirmed not auto-restarting (chkservd disarmed)
- ClamAV signature DB current and daemon running
- 0 infections detected in `/home`

Phase 6: Production cutover and what surfaced

The cushion server went live during Phase 6, with NSG opened, DNS cut over, and certificates issued. Several long-buried surprises surfaced. Each one would have been a painful production outage on the new infrastructure if discovered later instead of now.

Stale internal-IP references all over the cPanel stack

The recovery VM has internal IP `<new-vm-ip>`, but the original VM had `<old-vm-ip>`. The pre-incident snapshot carried the old IP into many places:

- `/var/cpanel/mainip` and `/etc/wwwacct.conf` (cPanel's stored "main IP", drives Apache vhost binding)
- `/var/cpanel/conf/apache/primary_virtual_hosts.conf` (Apache vhost-IP map)
- `/var/cpanel/users.cache/*` (cPanel per-user JSON snapshots, held the old IP in `"IP":"<old-vm-ip>"`)
- `/var/cpanel/databases/*.json` (MySQL grant cache)
- `/etc/nginx/conf.d/*` (upstream server addresses AND variable-reference suffixes encoding the old IP into the variable name)
- `/etc/apache2/conf/httpd.conf` (auto-generated from the above; rebuilds reverted my fixes until I addressed all source files)
- `/etc/hosts` (`<old-vm-ip> <server-internal-hostname>`; caused k3s `--node-ip` resolution to fail with "failed to find interface with specified node ip", crash-looping for 443 restarts)

Each of these had to be updated, and a rebuild order had to be respected. `/scripts/rebuildhttpdconf` regenerates Apache config from cPanel's stored data, then `/scripts/ea-nginx config --all` regenerates nginx from the new Apache state. If any source file still held the old IP, the rebuild would re-introduce it.

CSF + Kubernetes iptables collision

CSF (the host firewall installed in Phase 1.5) flushes iptables on activation/restart. Kubernetes' `kube-proxy` maintains its own iptables NAT chains (`KUBE-SERVICES`, `KUBE-NODEPORTS`, `KUBE-EXT-*`) for service routing. CSF's flush wiped them, and CoreDNS couldn't reach the k8s API to resolve service names, making API pods crash-loop with `forward host lookup failed` errors when trying to connect to `<application-database>`.

Fix: add the k3s pod CIDR and service CIDR (k3s defaults) to `/etc/csf/csf.allow`, then restart k3s **after** CSF has settled. Order matters. Kube-proxy rebuilds chains on startup, so it has to be the last thing to run.

Expired GoDaddy wildcard cert across all subdomains

cPanel's prior AutoSSL had configured all `*.<corporate-domain>` subdomains to share a single GoDaddy wildcard cert. That cert expired April 2, 2026, *before the encryption event of April 30*, so the snapshot carried an already-expired cert. Browsers visiting any subdomain would have shown a cert-expired error.

Fix: issued per-domain Let's Encrypt certs via `certbot --webroot` (not `--apache`, since cPanel uses `httpd` not `apache2ctl`), then manually replaced each `/var/cpanel/ssl/apache_tls/<domain>/combined` with the new LE cert content. Apache's `apachectl graceful` was insufficient to re-read the new cert files; `systemctl restart nginx` was required for the new certs to actually be presented on TLS connections.

Final cert state: 7 valid LE certs covering all served subdomains, each expiring between June 7 and July 29, 2026, with certbot's systemd timer scheduled for auto-renewal.

Pre-existing 502 on every subdomain

Independent of the above, every served subdomain had been returning **502 Bad Gateway** since the recovery VM was first provisioned. Nginx upstream pool was hard-wired to `<old-vm-ip>:444` (old VM internal IP). This had been silently broken for the entire period the recovery VM existed. Catching it now, while we still had Apache snapshots and the working data extraction confirmed clean, was much cheaper than catching it during a customer-facing cutover.

Cumulative time saved by catching these in remediation, not production

Surprise	If found in production
Stale internal IP across 6 cPanel files	Multi-hour outage; would have to debug cPanel rebuild order live
`/etc/hosts` k3s node-ip mismatch	Total k3s outage; `<primary-product>`/`<staging-environment>` down
CSF wiping kube-proxy iptables	All k8s services unreachable; pods crash-looping with cryptic DNS errors
Expired GoDaddy wildcard cert	All subdomains showing cert-expired error in

	browsers; emergency cert reissue under load
nginx upstream pinned to old VM IP	Every subdomain 502 on first customer hit

Every one of these was a multi-hour production fire if encountered cold. Caught and resolved in the cushion phase, they cost minutes apiece.

Outcome and current state

The migration cushion has been operational since April 30, 2026: production and staging applications serving real customer traffic via k3s; PHP applications responding; static sites delivering content; all HTTPS via valid Let's Encrypt certs.

The longer-term migration tracks are proceeding per plan in parallel.

Customer notification was drafted in the initial six-hour window and reviewed by counsel before going out; affected segments received tailored variants per their notification obligations. Forensic deep-dive on the original encrypted VM remains deferred. Snapshots are preserved indefinitely; a future log review of `/usr/local/cpanel/logs/access_log*` and `/var/cpanel/logs/login_log*` for unusual auth events in the April 28-30 window, cross-referenced against the indicators in CVE-2026-29201/202/203, can definitively confirm or rule out the cPanel-vuln entry-vector hypothesis whenever it's commissioned.

Hardened-state snapshots of the cushion server are in place for rollback safety. CSF, cPHulk, ModSecurity (with OWASP CRS V3), and WordFence dashboards are all monitored.

Why we wrote this

This case study is published to demonstrate that **AI-orchestrated incident response, when run by a persistent-identity agent on top of disciplined backups, turns ransomware from an existential threat into a recoverable event**. The AI layer is what made the difference. The backup layer is the substrate that let the AI layer matter.

- An AI agent without clean backups → nothing to recover from → the only outcome is paying the ransom or losing the data
- Clean backups without an AI agent → operator overwhelm in the high-stakes window → mistakes → partial recovery, missed indicators, longer downtime

What this incident shows is that the AI agent took the highest-stakes parts of the work (the parts where human cognitive load is highest and mistakes cost the most) and ran them with checklist discipline through a 6-hour continuous session. That's what reduced what could have been a multi-week recovery, or a forced ransom payment, to a single working day with zero data loss and zero customer data exposure.

For organizations considering similar architecture: the principles are not exotic. Multiple backup layers, separated storage substrates, version control for code, and a MindStone agent that is *itself* persistent enough to maintain coherent IR workflow across the long incident response timeline. None of those require enterprise budgets. They require discipline.

About the platform

The MindStone agent described in this case study runs on **MindStone**, an open-source persistent-identity AI agent platform. MindStone gives an AI agent a name, continuous memory across sessions, and SCRI-style semantic recall (Semantic Context Resonance Injection: weighted, experience-aware memory retrieval), so the agent doesn't reset between conversations and accumulates judgment over time.

- **MindStone:** github.com/R1ngZer0/MindStone, the platform
- **MindStone for Claude Code (MS4CC):** github.com/R1ngZer0/mindstone-for-claude-code, one of the substrate editions of MindStone
- **Synapse:** github.com/R1ngZer0/synapse, the cross-substrate comms service that lets MindStone agents on different runtimes coordinate in shared channels
- **SCRI paper draft**, the architecture document describing the resonance function, the three-tier memory, and the empirical evidence behind persistent-identity agents (available on request pending public release)

The MindStone marketing site at **mindstoneagent.ai** has install instructions, getting-started guides, and the longer architecture explanation.