

Sleeper Shells: How Attackers Are Planting Dormant Backdoors in Ivanti EPMM

F defusedcyber.com/ivanti-epmm-sleeper-shells-403jsp

Published on 09 February 2026

7 min read

Threat Intelligence

Vulnerability Advisory

Ivanti

SLEEPER SHELLS:

How Attackers Are Planting Dormant Backdoors in Ivanti EPMM



Exploitation of Ivanti Endpoint Manager Mobile (EPMM) has been relentless since vulnerability disclosure. That's not necessarily news. Major institutions - governments included - have already been compromised through this vector, and we're tracking another exploitation wave as it develops.

On **February 4th, 2026**, a coordinated campaign started across our telemetry with a differing pattern to previous mass exploitation. Rather than the smash-and-grab post-exploitation you'd expect - dropping traditional webshells, running recon and enumeration commands - this operator did something more deliberate, uploading a payload, confirming it landed, and leaving.

No commands were executed, the implant was simply left in place.

Key Takeaway: This campaign deployed a dormant in-memory Java class loader to `/mifs/403.jsp` - a somewhat lesser common webshell path. The implant can only be activated with a specific trigger parameter, and no follow-on exploitation has yet been observed. This is suggestive of initial access broker (IAB) tradecraft: gain a foothold, then sell or hand off access later.

The Vulnerabilities

Ivanti disclosed two critical vulnerabilities in EPMM - CVE-2026-1281 and CVE-2026-1340 - both covering authentication bypass and remote code execution, affecting different packages (aftstore and appstore respectively). The practical outcome is the same: unauthenticated access to application-level endpoints. Ivanti published patching guidance via their [security advisory](#), and exploitation in the wild followed shortly after.

Most of the early activity was predictable - opportunistic scanning, mass exploitation, and commodity webshell drops.

The 403.jsp Campaign

Every exploit from this campaign dropped a webshell to the path:

```
/mifs/403.jsp
```

The path itself isn't new - 403.jsp has been observed in prior compromises targeting Ivanti and MobileIron infrastructure, typically alongside other JSP-based webshells. What made this campaign worth writing about isn't where the payload was placed, but what it did once it got there.

Rather than deploying a conventional webshell with interactive command execution, the operator delivered a Base64-encoded Java class file via HTTP parameters. Each payload decoded to valid Java bytecode (identifiable by the CAFEBAFE header) functioning as a dormant in-memory class loader - not an immediately usable backdoor. The distinction matters, and we'll get into why below.

Inside `base.Info` - The In-Memory Loader

The payload is a compiled Java class (`base.Info`, source file `Info.java`). It's not a webshell in the traditional sense - it doesn't provide command execution, file access, or any interactive capability on its own. It's a stage loader: its only purpose is to receive, load, and execute a second Java class delivered via HTTP at a later time.

How It Works

The class uses `equals(Object)` as its entry point, which is unusual but intentional - standard servlet handler methods like `doGet` or `doPost` are more likely to be flagged by security tooling. The `parseObj` method extracts `HttpServletRequest` and `HttpServletResponse` from the argument, with fallback logic to handle `PageContext` objects and various servlet wrapper/facade patterns. This makes the loader portable across different container environments.

Once invoked, the loader checks for an HTTP parameter named `k0f53cf964d387`. If present, it strips a two-character prefix from the value and Base64-decodes the remainder into raw bytes. These bytes are then loaded as a Java class in memory via a reflective call to `ClassLoader#defineClass` - nothing is written to disk. The resulting class is instantiated with a string argument containing basic host information, and its `toString()` output is returned in the HTTP response.

The response body is wrapped in fixed delimiters (`3cd3d / e60537`) and served as `text/html`, making it straightforward for automated tooling to parse.

For Base64 handling, the loader supports both `java.util.Base64` (Java 8+) and `sun.misc.BASE64Decoder`, ensuring compatibility across older and newer JVM versions.

Host Fingerprinting

Before invoking the second-stage class, the loader gathers a small set of environment details: the application working directory (`user.dir`), filesystem root paths (drive letters on Windows, `/` on Unix), the OS name, and the running username. This information is passed to the second-stage class as a constructor argument - likely to help the operator orient themselves on the target at a later date.

What We Didn't See

This is the part that matters from a defensive standpoint. Across all observed telemetry, the loader was deployed and confirmed functional, but **no follow-on requests were observed supplying a second-stage class to the trigger parameter**. The access was established and then left dormant.

This is consistent with initial access broker (IAB) activity. The tooling is generic and container-agnostic - it's built to work reliably across environments, not to perform any specific post-exploitation task. The operator deployed it at scale, verified it landed, and moved on. The likely purpose is to package confirmed, working access for handoff or sale to a separate party, who would activate it from different infrastructure at a later time.

That separation - one actor establishes access, another exploits it - is what makes this pattern difficult to detect in practice. There's a gap between the initial compromise and eventual use where the telemetry trail simply goes quiet.

What To Look For

If you're running Ivanti EPMM, the presence of any of this activity should be treated as **evidence of compromise or attempted compromise** - even without follow-on exploitation. *Especially* without follow-on exploitation. The absence of further activity doesn't mean the access isn't valuable; it may simply mean it hasn't been activated yet.

Immediate actions: Patch Ivanti EPMM per vendor guidance immediately. Restart affected application servers to flush in-memory implants - this is critical, as the payload never touches disk. Then review access logs with the indicators below.

Log Indicators

- Any requests to `/mifs/403.jsp`
- Large Base64 parameters beginning with `yv66vg` (the Base64 encoding of the `CAFEBABE` Java magic bytes)
- The parameter name `k0f53cf964d387` in request strings
- Response bodies containing the markers `3cd3d` or `e60537`
- Any response containing `ERROR://` (the loader's error format)

Indicators of Compromise

Artifact

Field	Value
Class Name	<code>base.Info</code>
Source File	<code>Info.java</code>
SHA-256	<code>097b051c9c9138ada0d2a9fb4dfe463d358299d4bd0e81a1db2f69f32578747a</code>

Network IOCs - Source IPs

IP Address	Organization	ASN	Country
104.219.171.96	Datacamp Limited	AS212238	
108.64.229.100	AT&T Enterprises, LLC	AS7018	

IP Address	Organization	ASN	Country
115.167.65.16	NTT America, Inc.	AS2914	
138.36.92.162	HOSTINGFOREX S.A.	AS265645	
146.103.53.35	Datacamp Limited	AS212238	
148.135.183.63	Datacamp Limited	AS212238	
151.247.221.59	Datacamp Limited	AS212238	
166.0.83.171	UK Dedicated Servers Limited	AS42831	
172.59.92.152	T-Mobile USA, Inc.	AS21928	
185.240.120.91	Datacamp Limited	AS212238	
185.239.140.40	Datacamp Limited	AS212238	
194.35.226.128	LeaseWeb Netherlands B.V.	AS60781	
193.41.68.58	LeaseWeb Netherlands B.V.	AS60781	
77.78.79.243	SPCom s.r.o.	AS204383	
62.84.168.208	Hydra Communications Ltd	AS25369	
45.66.95.235	Hydra Communications Ltd	AS25369	
46.34.44.66	Liberty Global Europe Holding B.V.	AS6830	

More IOCs available in Defused telemetry.

Third-Party Detection

The **base.Info** class was submitted to VirusTotal, where it received a hit from Nextron Systems' THOR APT Scanner under a generic JSP webshell characteristics rule. Traditional AV engines largely miss the artifact - unsurprising for a payload that never touches disk - but heuristic engines correctly identify the behavioural signatures consistent with in-memory class loaders.

See the [VirusTotal entry here](#).

The Quiet Ones Are Worse

There's a tendency in incident response to prioritise the loud compromises - the ransomware detonations, the mass data exfiltration, the lateral movement storms that light up every detection rule in the stack. This campaign is a useful reminder that the most dangerous intrusions are often

the ones that don't do anything. Yet.

The use of undocumented JSP paths, in-memory Java loaders, and dormant execution triggers represents a meaningful step forward in post-exploitation tradecraft targeting enterprise mobility infrastructure. When the operator's goal is to create *inventory* rather than cause immediate damage, all the usual urgency signals go quiet - and that's exactly the point.

If you're running Ivanti EPMM, don't wait for the second act. Patch, restart, and hunt. The loader is patient. You don't have to be.

Parts of the technical analysis above have been AI-assisted.