TRYHACKME / PHP2SHELL / FIELD NOTES

PHP2Shell Walkthrough: I Turned a Cookie Into a Shell

A progress bar, an out-of-memory bailout, and a session cookie form an unexpected path to code execution. Here is what I learned across all eight tasks of the TryHackMe room.



By **Nazline Mwita** · 30 October 2026 · 9 min read · Authorized lab

01 / REACH	02 / BAILOUT	03 / LEAK	04 / EXECUTE	05 / DEFEND
Upload progress	**Memory limit**	**Set-Cookie header**	**Cleanup pointer**	**Patch PHP**

The most striking part of this room is how little the application itself has to do. The lab page prints **"OK."** The vulnerable behavior happens inside the PHP runtime before the page's own code runs. I completed the room's eight tasks in a controlled TryHackMe environment and traced how an ordinary upload-progress feature becomes a memory leak and then command execution.

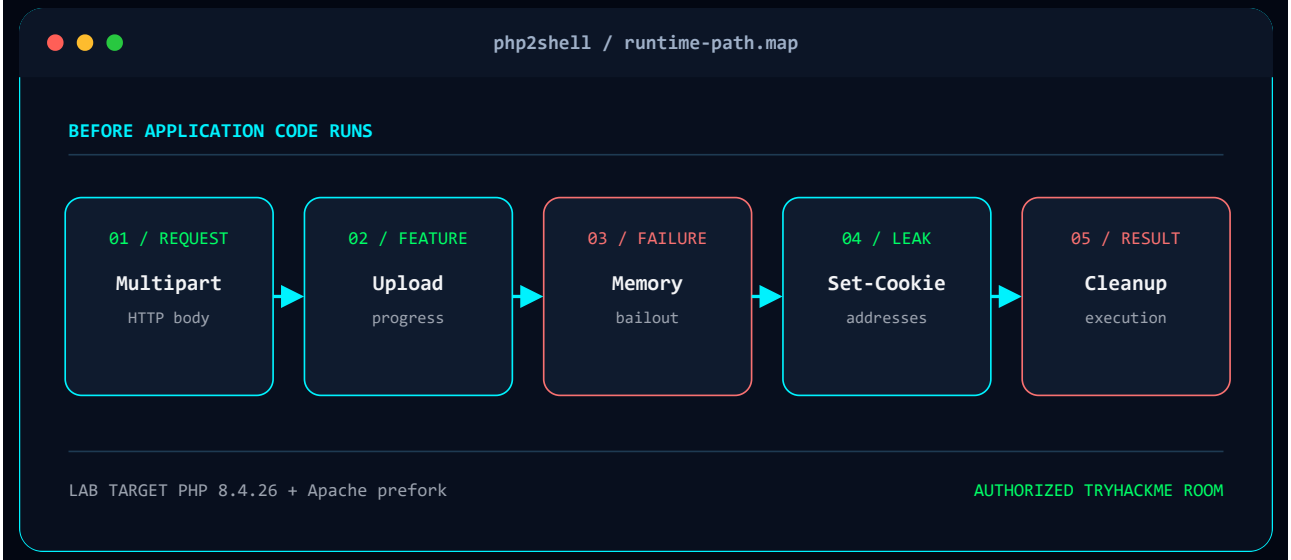


Figure 1. The PHP2Shell lab chain, from early request processing to code execution.

> **The central lesson:** a clean application does not remove risk in the runtime beneath it. PHP's own security advisory calls for a runtime update, even when the application code is sound.

1. Introduction: the bug is below the website

The room covers **CVE-2026-109618** and **CVE-2026-109619**, disclosed in October 2026. PHP's advisory says these critical flaws can permit unauthenticated remote code execution. The lab makes that abstract warning concrete: its target runs PHP 8.4.26, one release before the 8.4.27 security fix, behind Apache in a container.

TryHackMe supplies SSH access to inspect the target and provides the exploit tools. That makes this a white-box learning exercise, even though the vulnerability itself can be reached by unauthenticated web requests.

2. A useful feature: upload progress

PHP can record a large upload's progress in the user's session so a page can display a progress bar. A multipart form field named `PHP_SESSION_UPLOAD_PROGRESS` enables that tracking when the feature is on. The work happens while PHP parses the request body, before the requested script executes.

That timing matters. The attacker reaches PHP's session upload-progress path without needing a login or a mistake in the site's application code. The lab uses this early request-processing window as its entry point.

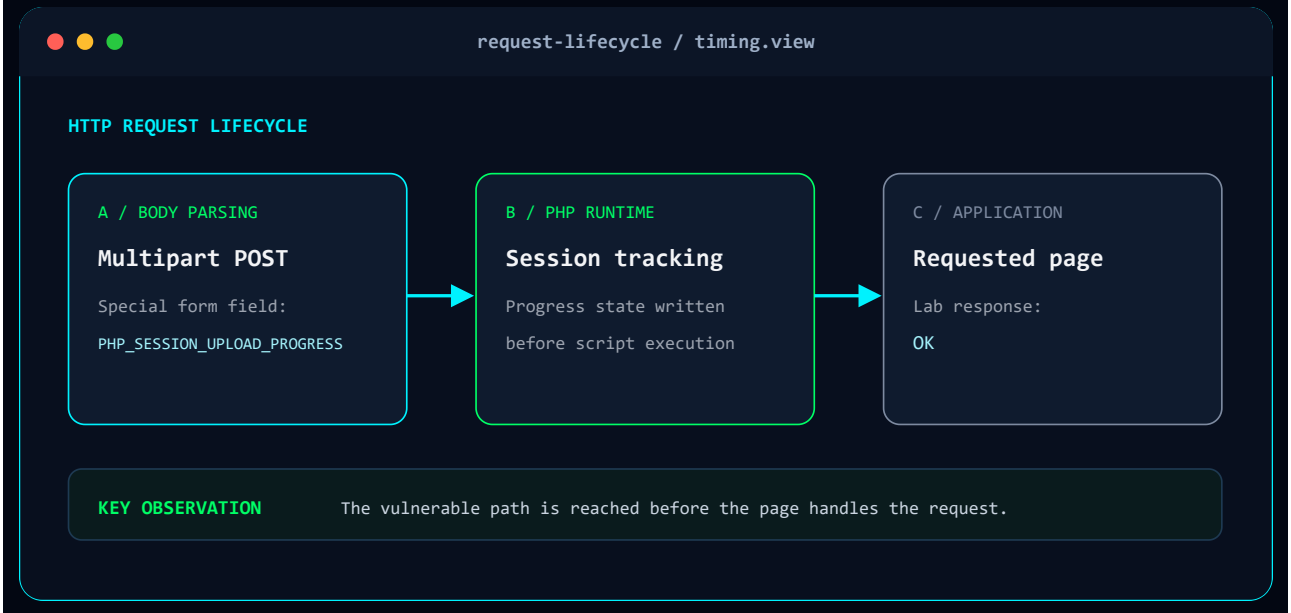


Figure 2. Upload-progress tracking runs during body parsing, before the lab page returns "OK."

ROOM CHECK / TASK 2

Default form-field name: `PHP_SESSION_UPLOAD_PROGRESS`.

---

3. The bailout at the wrong moment

Every PHP request has a memory limit. When the request exceeds it, PHP performs an emergency **bailout**. In the vulnerable path, request cleanup leaves internal state referring to memory that has already been freed. A later request on the same worker can encounter that stale state before normal initialization resets it.

In simple terms, PHP lets go of a piece of memory but retains a reference to it. The room uses **CWE-824** to describe this uninitialized-pointer weakness. The important effect is the boundary crossing between one request and the next.

ROOM CHECK / TASK 3

Emergency jump: bailout. **Weakness:** CWE-824.

---

4. Recon: an “OK” page and a vulnerable worker

From the supplied SSH shell, I identified the `php:8.4.26` container and checked the conditions the room was built around: Apache prefork with mod_php, upload progress enabled, one worker, and a low memory limit. The index page responded with a single word: **OK**.

This was the useful recon result. There was no application upload page to exploit. The attack path was in PHP's processing of the incoming request, below the site.

ROOM CHECK / TASK 4

Target PHP version: 8.4.26. **Index response:** OK.

---

5. A cookie that discloses memory

After the bailout, the next request can reuse freed space while the stale session pointer still exists. PHP may then interpret the data at that address as a session identifier. When it writes a `Set-Cookie` response header, bytes from the worker's memory can appear in the cookie value.

The lab's exploit uses that disclosure to identify a heap address and then the PHP library base address. These address leaks weaken ASLR, which normally makes code and memory locations unpredictable. In the room, the leak is the bridge between the stale pointer and the later control-flow step.

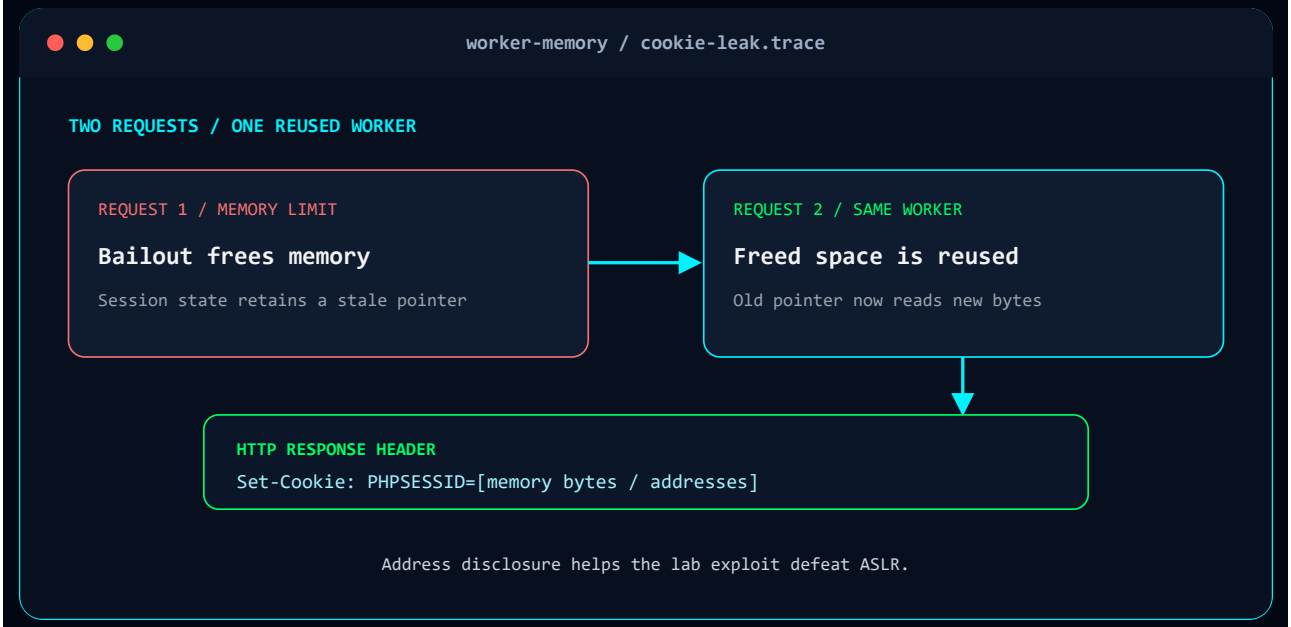


Figure 3. A reused worker can expose memory through the session cookie response.

ROOM CHECK / TASK 5

Response header carrying the leak: `Set-Cookie` .

---

6. From address leak to command execution

With the needed addresses known, the lab exploit places a forged internal PHP hash-table object into the reused memory. Its cleanup function pointer is made to point at a command-execution function. When PHP tidies up at the end of the request, the unexpected pointer redirects that cleanup call.

I ran the provided lab exploit and observed its stages: worker rotation, heap disclosure, library disclosure, the forged object, and command execution. The `id` proof showed execution as **www-data**, the web-server account. A later command returned the room flag.

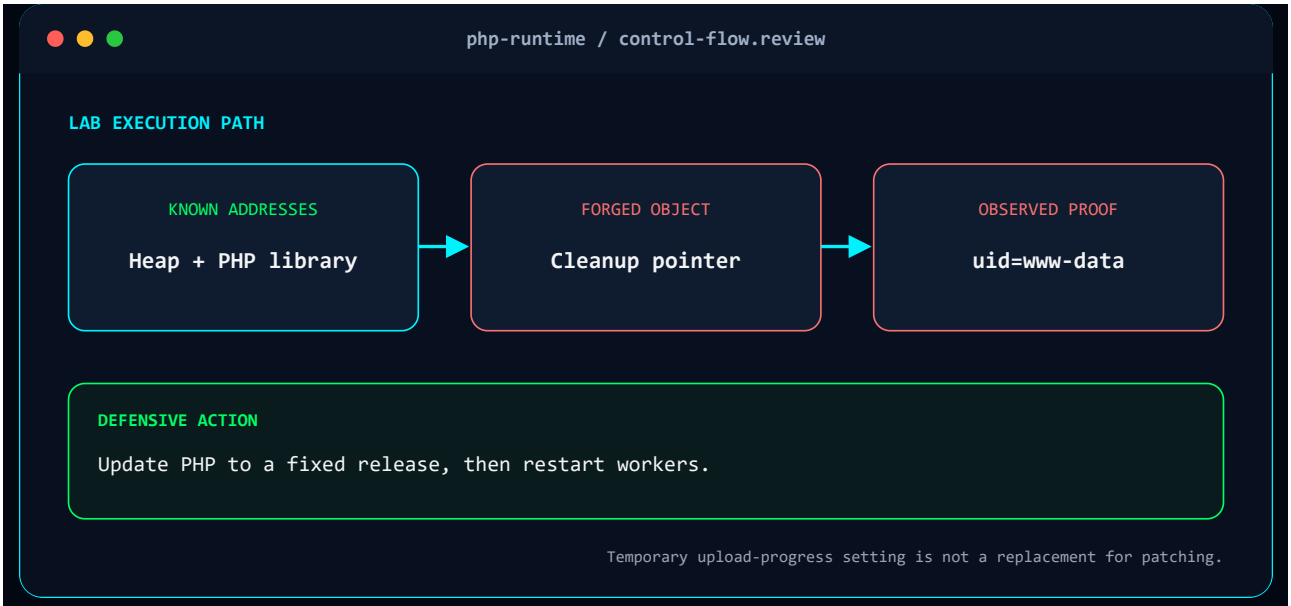


Figure 4. The lab's execution proof and the defensive action that breaks the chain.

ROOM CHECK / TASK 6

Execution context: `www-data`. The room flag was returned by the supplied lab exploit; it is omitted here so readers can complete the exercise themselves.

7. Detection and the real fix

Updating a CMS, plugin, or application framework does not patch the affected PHP runtime. PHP's security advisory lists the fixed releases below and says all PHP installations serving web requests should be updated. The advisory also notes that distributors may provide backported fixes, so verify the installed vendor package rather than relying only on a version banner.

PHP branch	Fixed release
8.5	8.5.12
8.4	8.4.27
8.3	8.3.36
8.2	8.2.35

Restart PHP workers after updating. While patching is underway, PHP recommends setting `session.upload_progress.enabled=0` as a temporary measure, followed by a worker restart. It does **not** replace the update. Investigate unusual multipart requests, memory-limit failures, worker instability, and abnormally large session cookies as supporting signals.

> ROOM CHECK / TASK 7
>
> **Fixed PHP 8.4 release:** 8.4.27. **Upload-progress setting:** `session.upload_progress.enabled`.

---

8. What I took away

A progress indicator opened the path. A memory-limit bailout left stale state. A cookie exposed addresses. Cleanup became code execution. Each step came from the runtime's behavior, while the application page did almost nothing.

That is the security lesson I am keeping from this room: include the runtime and its deployment mode in the attack-surface inventory, patch it with the same urgency as application code, and test any claim of protection against the stage where the vulnerable work actually happens.

Sources: PHP security advisory, 21 and 23 October 2026 · My original Medium walkthrough. All hands-on observations above are from the authorized TryHackMe lab.