

What are 2Shell vulnerabilities?

A 2Shell (or 4Shell) vulnerability is a nickname for a remote code execution flaw in a component so widely deployed that one bug exposes a large share of the internet at once. The name reads as "from X to a shell": input reaching PHP, Node.js, React, Log4j or Bash ends with the attacker running commands on the server. Members include PHP2Shell (CVE-2026-109618/109619), Node2Shell and HTTP2Shell (CVE-2026-111497), React2Shell (CVE-2025-55182), Log4Shell (CVE-2021-44228), Spring4Shell, Text4Shell, ProxyShell and Shellshock.

HOW IT WORKS

01 The family, at a glance

- PHP2Shell (CVE-2026-109618, CVE-2026-109619, October 2026): memory corruption in PHP core during request setup, reachable before any application code runs.
- Node2Shell (CVE-2026-111497, October 2026): an out-of-bounds write in the Brotli decoder bundled with Node.js, reached by Brotli-compressed request bodies or responses.
- HTTP2Shell (CVE-2026-111497): the same Brotli bug seen across every HTTP stack that bundles the vulnerable decoder, including Python packages.
- React2Shell (CVE-2025-55182, December 2025): insecure deserialization in React Server Components, hitting Next.js and other RSC frameworks.
- Log4Shell (CVE-2021-44228, December 2021): a logged string triggers a JNDI lookup in Apache Log4j 2 and loads remote code.
- Spring4Shell (CVE-2022-22965, March 2022): Spring data binding reaches the class loader and writes a web shell through Tomcat logging.
- Text4Shell (CVE-2022-42889, October 2022): Apache Commons Text string interpolation evaluates script expressions.
- ProxyShell (2021) and ProxyNotShell (2022): chained Microsoft Exchange flaws ending in code execution.
- Shellshock (CVE-2014-6271, 2014): Bash runs commands appended to function definitions in environment variables.

SOURCES

- [1] MITRE ATT&CK: Exploit Public-Facing Application (T1190)
- [2] CISA Known Exploited Vulnerabilities Catalog
- [3] NVD: CVE-2026-109618 (PHP2Shell)
- [4] NVD: CVE-2026-111497 (Node2Shell)

02 What the family has in common

Different bugs, same shape. Each one combines four traits that make a flaw spread faster than teams can patch:

- Ubiquity. The component sits under a large share of web servers: PHP runs most sites with a known server-side language, and Node.js, React, Log4j and Bash are just as common in their worlds.
- Pre-authentication reach. The vulnerable code runs while the request is being parsed, logged or decompressed, before login checks or application logic.
- Hidden dependency. Teams often do not know they run it. Log4j sat four levels deep in Java dependency trees; Node.js and Brotli ship inside container images and single binaries that a package scan does not see.
- A race against public exploitation. PHP2Shell was probed four days before disclosure, and Log4Shell and React2Shell were mass-exploited within days of their advisories.

03 The root-cause classes behind the names

Strip the branding and the family falls into three classic bug classes:

- Memory corruption in native code: PHP2Shell (out-of-memory handling and stale state in the C engine) and Node2Shell/HTTP2Shell (an out-of-bounds write in the Brotli C decoder). See image-based heap overflows for how a decoder write becomes code execution.
- Unsafe deserialization or expression evaluation: React2Shell (React Flight payloads), Log4Shell (JNDI lookups), Text4Shell (interpolators) and Spring4Shell (data binding into class-loader properties). These are cousins of insecure deserialization and template injection.
- Injection into a shell or privileged backend: Shellshock (Bash) and ProxyShell (Exchange PowerShell), which map to OS command injection.

Knowing the class tells you what an edge rule can and cannot block. Expression bugs have recognizable strings like `${jndi:-`

; memory-corruption bugs hide inside valid-looking compressed or multipart bodies.

04 How to respond when the next one lands

The playbook is the same every time:

- Inventory what actually runs. Check the runtime inside every deployed image and host, not just the manifest: `php -v`, `node -v`, `java -version` and your software bill of materials. Bundled runtimes are where these flaws hide.
- Apply the edge mitigation from the official advisory while you patch, for example blocking unused `multipart/form-data` for PHP2Shell or rejecting `Content-Encoding: br` for Node2Shell. Treat it as a stopgap, never the fix.
- Patch and rebuild. Upgrade the component, then rebuild containers and redeploy so the old binary is gone.
- Hunt for prior compromise. If exploitation started before your patch, look for new processes spawned by the web server user, unexpected outbound connections and web shells dropped in web roots.
- Validate. Retest the exposed paths to confirm the fix holds, ideally with a tester who can prove exploitability rather than match a version string.

Find the path to a shell before an attacker does.

securelayer7.net/learn/web-exploitation/what-are-2shell-vulnerabilities

[Open online](https://securelayer7.net/learn/web-exploitation/what-are-2shell-vulnerabilities)