What is Application Security

2026-06-20

Application security: SQL Injection, Log4Shell, OWASP Top 10, and Input Validation — attacks that affect confidentiality and integrity, not availability.

The third part of the (D)DoS series — but a step sideways: this post covers attacks at the Application layer that affect **confidentiality and integrity** rather than availability. SQL Injection, Log4Shell, and similar vulnerabilities have a different threat model than the DDoS attacks covered in the technical part.

Series navigation: - Part 1 — Non-technical: Business, Social, Informational - Part 2 — Technical: Network floods, BGP hijacking, Layer 7, Resilience - **Part 3 — Application Security (this post)**

Introduction

The Application layer is the level where most services expose their functionality — SMTP/IMAP (E-Mail), HTTP, PostgreSQL client protocol, and many more protocols live here.

This level is often described as the weakest — though as we'll see, that framing is incomplete — because at this level "everybody" wants to be part of it.

The Business Layer is the reason why all the setup exists. Due to that fact there are business requirements or ideas which come from everyday usage of the application, there are always security requirements, and the used technologies, libraries and tools get newer versions and should be updated. The big discussion will be done for the business requirements but the discussion for updating the used libraries will be short or not even done because only a small group of people have the knowledge for deeper discussions. This leads to the fact that libraries and tools are very seldom updated — which opens the door for vulnerabilities like the widely known Log4J Vulnerability CVE-2021-44228.

The Attack Surface

{{ bounded_image(src="/img/ddos-application-attack-surface.drawio.png", alt="Application Attack Surface: classical attacks (SQL Injection, XSS, Log4Shell) and AI-era extensions (Prompt Injection, Agent Tool Abuse), with Libraries and Dependencies as the shared root cause", max_width=960) }}

The classical attack surface — SQL Injection, Cross-Site Scripting, insecure deserialization, and the broader range of input-based attacks — has been well-documented for years. To address these risks, security should be built in from the beginning of the development, test, and run process.

There is a group which has specialized in web security called the Open Worldwide Application Security Project (OWASP), and their Top Ten list is a good starting point for understanding the most common unmitigated security risks.

AI-Assisted Vulnerability Discovery

The attack surface described above was always exploitable — what kept it tolerable was an implicit time buffer: obscure latent bugs took years to surface. Log4Shell had existed in the codebase since 2013 and went unnoticed in millions of deployments until 2021. That buffer is gone.

Worth noting: the examples below are not typical web application bugs — they include SQLite, OpenSSH, GnuTLS, and low-level C libraries with millions of fuzzing-hours behind them. The honest 2026 picture is not that the Application layer is uniquely weak, but that latent bugs exist at every layer and the cost of finding them has collapsed. These are real CVEs in widely deployed software, not research prototypes.

Google Big Sleep (DeepMind + Project Zero) is the most prominent example. In late 2024 the agent found a stack buffer underflow in SQLite — described by Google as the first public case of an AI agent discovering a previously unknown, exploitable memory-safety issue in production software. In mid-2025 Big Sleep found CVE-2025-6965, a memory corruption vulnerability in SQLite versions before 3.50.2, which was exclusively known to attackers and on the verge of active exploitation. This is considered the first case of an AI agent stopping an active threat before it was used.

OpenAI Aardvark / Codex Security scanned more than 1.2 million commits over 30 days and found 792 critical findings and over 10,000 high-severity issues — including in OpenSSH, GnuTLS, libssh, PHP, and Chromium. The Thorium browser yielded CVE-2025-35430 through CVE-2025-35436 from that scan.

DARPA AI Cyber Challenge (AIxCC) ran autonomous systems against each other in a public competition. Team Atlanta's "Atlantis" found six zero-days in SQLite3 and patched one of them autonomously during the event. After the finale in August 2025, the teams turned their tools on real critical open-source packages.

XBOW became the first non-human participant to reach #1 on HackerOne and identified over 1,000 vulnerabilities at companies including AT&T, Epic Games, Ford, and Disney.

Anthropic reported that Claude found and validated over 500 high-severity zero-days in production open-source codebases.

For self-hosted use, Vulnhuntr (Protect AI) is an open-source static analyzer for Python codebases that uses Claude to find zero-days. False-positive rate is a known challenge in this class of tools — the more rigorous systems run findings through a triage chain before reporting.

The Window Is Shrinking

The tools above compress that window significantly: AI systems now scan millions of commits in days, find memory-safety issues in production codebases, and in some cases generate and validate patches autonomously.

The implication for defenders is not that AI will solve security — it won't. But the assumptions behind slow, deferred patching cycles no longer hold. Vulnerabilities that would have taken years to surface are now found in weeks.

> ⚠ **The window is shrinking**
>
> The window between a vulnerability existing and being discovered is shrinking. "Keep dependencies updated" is no longer just good hygiene — it is the primary mitigation against a class of threats that is being uncovered faster than most organizations patch.

Protection

Input Validation

From my point of view, one of the most effective protections is Input Validation at the earliest possible point in the application. The difficult part is defining what constitutes valid input for the specific application — this requires understanding the business logic, not just applying generic filters.

Web Application Firewall (WAF)

A Web Application Firewall (WAF) can be a valid protection layer — but from experience, a lot of organizations deploy a WAF in front to have the security ☒ without configuring it to actually protect the specific application. Default rules cover known patterns; custom application behavior requires custom rules.

Keep Dependencies Updated

From my point of view one of the most important and easiest mitigations — and from experience the most unwanted in enterprise setups — are frequent updates of the used libraries and tools. Log4Shell is a good example of how a vulnerability in a widely used library can stay unpatched for years because update processes are not in place.

Conclusion

Application security is a broad field that goes far beyond DDoS protection. The attacks here primarily threaten confidentiality and integrity — what data can an attacker read or modify — rather than availability. Both dimensions matter; they just require different mitigations and different conversations within an organization.