# **Common Application Security Vulnerabilities**

| **Attack Type** | **Cause (Root Problem)** | **Impact (What Attacker Gains)** | **Mitigation (Defense)** |
| --- | --- | --- | --- |
| **Buffer Overflow** | Writing more data than memory buffer can hold (lack of bounds checking). | Program crash, arbitrary code execution, privilege escalation. | Use safe languages (Java, Python), bounds checking, DEP/ASLR, stack canaries. |
| **Cross-Site Scripting (XSS)** | Unsanitized user input is sent to the browser and executed as JavaScript/HTML. | Cookie/session theft, redirecting users, defacement, full account takeover. | **Input validation**, **Output encoding**, Content Security Policy (CSP), HttpOnly/Secure cookies. |
| **Command Injection** | Unsafe user input is passed into system shell or API call. | Remote code execution, system compromise, data loss. | Whitelist input, parameterized APIs, avoid shell calls with raw input. |
| **Path Traversal (Directory Traversal)** | Application trusts file paths from user input (e.g., ../../etc/passwd). | Reading sensitive system files, data exfiltration, config leaks. | Input validation, canonicalize paths, restrict to sandboxed directories, use safe file APIs. |
| **Output Encoding** *(defense technique, not an attack)* | — | Prevents malicious user input from being executed by the browser (e.g., <script> shown as text instead of running). | Encode special characters (< → &lt;, > → &gt;), always applied before rendering user input in HTML/JS/URL. |

**Key Insight:**

* **Input Validation** = *Don’t trust what comes in* (clean or reject bad input).
* **Output Encoding** = *Don’t trust what goes out* (make sure data is displayed as text, not executed).

Together, they block most **XSS, injection, and traversal attacks**.