Reflected XSS into HTML context with nothing encoded occurs when an application takes user-supplied input (e.g. a ?q= parameter) and injects it directly into an HTML page without any output encoding/escaping. Because the input is placed raw into an HTML text node or element, an attacker can craft a URL that includes <script> (or other JS-bearing payloads) which the browser will execute when a victim opens the link.

Quick example (vulnerable pattern)

```
<!-- server returns this with user input inserted raw -->
<h1>Search results for: USER_INPUT</h1>
```

If USER_INPUT contains <script>alert(1)</script> and the application does **no encoding**, the script runs in the victim's browser.

Mpact

Reflected XSS allows arbitrary JavaScript execution in the context of the target site (same-origin). Typical impacts:

- Session theft (if cookies are not HttpOnly)
- Account takeover via CSRF-like flows or automated actions
- UI redressing / phishing inside the trusted site
- Data exfiltration or loading malicious assets

Severity is usually **High** because exploitation is remote and immediate.

Safe PoC (only on authorized/test systems)

```javascript
GET /search?q=<script>alert('xss')</script>
```

If the page reflects the parameter unescaped and alert('xss') executes, you have a reflected XSS.

Remediation (practical)

- **Always use context-aware output encoding**:
    - HTML text node → htmlspecialchars($input, ENT_QUOTES, 'UTF-8') (PHP) or equivalent.
    - Attribute context → attribute encoding (ENT_QUOTES).
    - JavaScript context → json_encode() or a proper JS-string encoder.
- Prefer secure templating engines that escape automatically (Twig, React JSX, etc.).
- Implement a Content Security Policy (CSP) to reduce impact (defense-in-depth).
- Set cookies to HttpOnly; Secure; SameSite where appropriate.
- Audit all reflection points: query params, POST body, headers (Referer, User-Agent).
