# CRLF Injection (Response Splitting)

Carriage Return Line Feed Injection

A web vulnerability where an attacker injects **(\r) and (\n) characters** into HTTP headers or responses, **manipulating the structure of the server's response**. It allows for:

* HTTP response splitting
* HTTP header injection
* Cross-site scripting (XSS)
* Web cache poisoning

# Why “CRLF”?

\r = Carriage Return (ASCII 13)  
\n = Line Feed (ASCII 10)  
Together (\r\n) = New line in HTTP protocol → separates headers and body

# How It Works

If a server uses unvalidated user input in headers:

response = HttpResponse()  
response["Location"] = user\_input

And attacker sends:  
 ?redirect=%0D%0ALocation: <http://evil.com>

The browser/server will interpret it as:  
HTTP/1.1 302 Found  
Location: /safe  
Location: <http://evil.com>  
  
Now the **malicious Location header** is injected, and browsers may follow it.

# Impact

| Impact Type | Description |
| --- | --- |
| HTTP Response Splitting | Split response into two → inject malicious headers or body |
| XSS via Header Injection | Inject script tags in custom headers or HTML |
| Web Cache Poisoning | Poison shared cache to store malicious versions |
| Session Fixation / Header Override | Inject Set-Cookie, Location, etc. |
| Content Spoofing | Fake response body with altered headers |

# Example:

<https://vulnerable.com/login?lang=en>  
**Malicious Input:** lang=en%0D%0ASet-Cookie: sessionId=malicious  
**Response becomes:**HTTP/1.1 200 OK  
Content-Type: text/html  
Set-Cookie: sessionId=malicious  
<html>...  
Session fixation or forced header change

# Types of CRLF Injection

**1. HTTP Response Splitting**

* **Definition**: Attacker injects \r\n to split one response into two.
* **Example**: Injected headers + malicious body
* **Mitigation**: Sanitize CRLF characters from user input.

**2. Header Injection**

* **Definition**: Malicious headers injected into response.
* **Example**: Set-Cookie, Content-Length, Location
* **Mitigation**: Never reflect user input in headers without encoding.

**3. Content Spoofing / XSS**

* **Definition**: Fake or altered page content delivered to user.
* **Example**: <script>alert('xss')</script>
* **Mitigation**: HTML encode and validate all user input.

**4. Web Cache Poisoning**

* **Definition**: Injected response is cached by reverse proxy or CDN.
* **Example**: Malicious headers cached and served to other users.
* **Mitigation**: Validate request parameters, set correct cache control.

# Detection Techniques

| Method | Tool | Description |
| --- | --- | --- |
| Manual Tampering | Burp Suite → Repeater | Inject %0d%0a, \r\n in query/headers |
| Curl Testing | Simple payloads |  |

curl [https://site.com/?input=abc%0d%0aX-Test: injected](https://site.com/?input=abc%0d%0aX-Test:%20injected)

| OWASP ZAP | Passive scanner flags unsafe header reflection |  
| PoC Page | Observe response split in headers or browser |  
| Observe Raw HTTP Response | In Burp or browser devtools (network tab)

# Mitigation

| Area | Fix |
| --- | --- |
| Input Validation | Strip or encode CR (\r) and LF (\n) characters from user input |
| Header Construction Libraries | Use high-level HTTP libraries that prevent manual header injection |
| Contextual Output Encoding | Encode all output according to HTTP, HTML, JSON, or URL context |
| Do Not Trust URL Parameters for Headers | Avoid reflecting raw query/input data in headers |
| Implement WAF Rules | Block special encoded characters like %0d, %0a in sensitive endpoints |
| Strict Response Formatting | Add extra security headers (e.g., X-Content-Type-Options, Content-Security-Policy) |

Sanitize input to remove \r and \n from headers.

Tools for CRLF Testing

| Tool | Use |
| --- | --- |
| Burp Suite | Manual injection, response inspection |
| OWASP ZAP | Header testing and alerting |
| CRLF Suite | Automated scanner for CRLF injection |
| curl | Raw HTTP requests |
| Postman | Header tampering (less effective for low-level testing) |

# Real-World Vulnerabilities

| Case | Impact |
| --- | --- |
| CVE-2016-8628 (Apache Tomcat) | HTTP response splitting |
| CVE-2021-23336 (Python urllib) | CRLF injection in redirect handling |
| Public CDN attacks | Cached poisoned response with CRLF-based headers |

# Points

“CRLF Injection abuses the fact that HTTP uses \r\n to separate headers — injecting these characters can break protocol structure.”

“It often leads to **HTTP Response Splitting**, which can enable **stored or reflected XSS**, cache poisoning, or session attacks.”

“During VAPT, Test headers like Location, Set-Cookie, and others using payloads such as %0d%0aX-Test: injected.”

# Payload Cheat Sheet

| Payload | Description |
| --- | --- |
| %0d%0aSet-Cookie: injected=value | Header injection |
| %0d%0aContent-Length: 0 | Response tampering |
| %0d%0aLocation: https://evil.com | Redirect manipulation |
| %0d%0a<script>alert(1)</script> | XSS via content injection |
| %0d%0aX-Test: CRLF-Test | PoC header injection |