U,{7c48c4de-3875-463f-b266-6b2a7d3cc62b}{63},3.125,3.125

* [Dashboard](https://academy.hackthebox.com/dashboard)
* [Modules](https://academy.hackthebox.com/modules)
* [Paths](https://academy.hackthebox.com/paths)

#### HTTP Attacks

1. [Page 6](javascript: void(0);)
2. CRLF Injection Prevention & Tools

# CRLF Injection Prevention

After seeing different ways to identify and exploit CRLF injection vulnerabilities, let's discuss how we can protect ourselves from these types of attacks. Afterward, we will briefly discuss tools that can help us identify CRLF injection vulnerabilities.

## Insecure Configuration

CRLF injection vulnerabilities can occur in any place where the CRLF control sequence has a special semantic meaning. As we have seen in the previous sections, this can include log files and particularly headers in HTTP and SMTP.

#### Log Injection

Let's start by looking at probably the most common form of CRLF injection vulnerability which is log injection. Consider this sample code snippet:

Code: php

function log\_msg($ip, $user\_agent, $msg) {  
 global $LOGFILE;  
 $log\_msg = "Request from " . $ip . " (" . $user\_agent . ")" . ": " . $msg;  
 file\_put\_contents($LOGFILE, $log\_message, FILE\_APPEND|LOCK\_EX);  
}  
  
$log\_msg($\_SERVER['REMOTE\_ADDR'], $\_SERVER['HTTP\_USER\_AGENT'], $\_POST['msg']);

The code snippet implements a custom log function that logs data to a log file. In particular, the log message contains user-supplied parameters such as the user agent and even data from an HTTP POST request. These user-supplied parameters can contain arbitrary characters, including a CRLF control sequence. As such, the above code is vulnerable to log injection which an attacker could exploit to forge log entries as discussed a couple of sections ago.

The most obvious way of preventing such issues is to use the logging functionality provided by the web server. However, if we need to implement a custom log functionality, we can avoid CRLF injection by ensuring that user-supplied input is always URL-encoded before its written to a file. In PHP, we can do so using the urlencode function. Since this function URL-encodes all non-alphanumeric characters including the CR and LF characters, this prevents CRLF injection issues.

#### Response Splitting

Similarly to preventing log injection issues, response splitting can be prevented by always using high-level functions for setting headers and cookies, especially if they contain user-supplied input. For instance, in PHP we can use the header function to set custom headers. While this function used to be vulnerable to CRLF injection, it has long been fixed since PHP 5.1.2, as we can see in the patch notes [here](https://www.php.net/releases/5_1_2.php):

HTTP Response Splitting has been addressed in ext/session and in the header() function.  
Header() can no longer be used to send multiple response headers in a single call.

The function rejects any input that contains the CRLF control sequence. However, this also means that PHP versions before 5.1.2 are potentially vulnerable to HTTP response splitting. If you ever encounter a system running such a long deprecated PHP version in a real-world engagement, it might be worth checking for HTTP response splitting.

An additional security measure is to URL-encode user input in headers and cookies. Particularly if the input contains a URL anyway, as was the case in the redirector service from a couple of sections ago.

#### SMTP Header Injection

SMTP Header injection is a variant of CRLF injection that can occur frequently due to the unawareness of PHP programmers, in particular, if the default functions provided by PHP are used to handle the sending of emails. Let's have a look at a vulnerable configuration:

Code: php

$to = "[recipient@demo.htb](mailto:recipient@demo.htb)";  
$subject = "You received a message";  
$message = "Here is the message:\r\n" . $\_POST['message'];  
  
$user\_mail = $\_POST['email'];  
$headers = "From: " . $user\_mail . "\r\n";  
$headers .= "Reply-To: webmaster@demo.htb\r\n";  
  
mail($to, $subject, $message, $headers);

The PHP function mail expects a recipient, a subject, a message body, and optionally a list of additional SMTP headers. For more details, have a look at the documentation [here](https://www.php.net/manual/en/function.mail.php). As shown in the code snippet above, additional SMTP headers can be supplied in a string that contains the headers separated by the CRLF control sequence. Since user-supplied input is used in the From header, the above code is vulnerable to CRLF injection.

To prevent this, we should URL-encode the user-supplied data before adding it to the SMTP headers to ensure that all CRLF characters are encoded:

Code: php

$to = "[recipient@demo.htb](mailto:recipient@demo.htb)";  
$subject = "You received a message";  
$message = "Here is the message:\r\n" . $\_POST['message'];  
  
$user\_mail = $\_POST['email'];  
$headers = "From: " . urlencode($user\_mail) . "\r\n";  
$headers .= "Reply-To: webmaster@demo.htb\r\n";  
  
mail($to, $subject, $message, $headers);

Generally, user-supplied input should not be used in SMTP headers when it is not necessary.