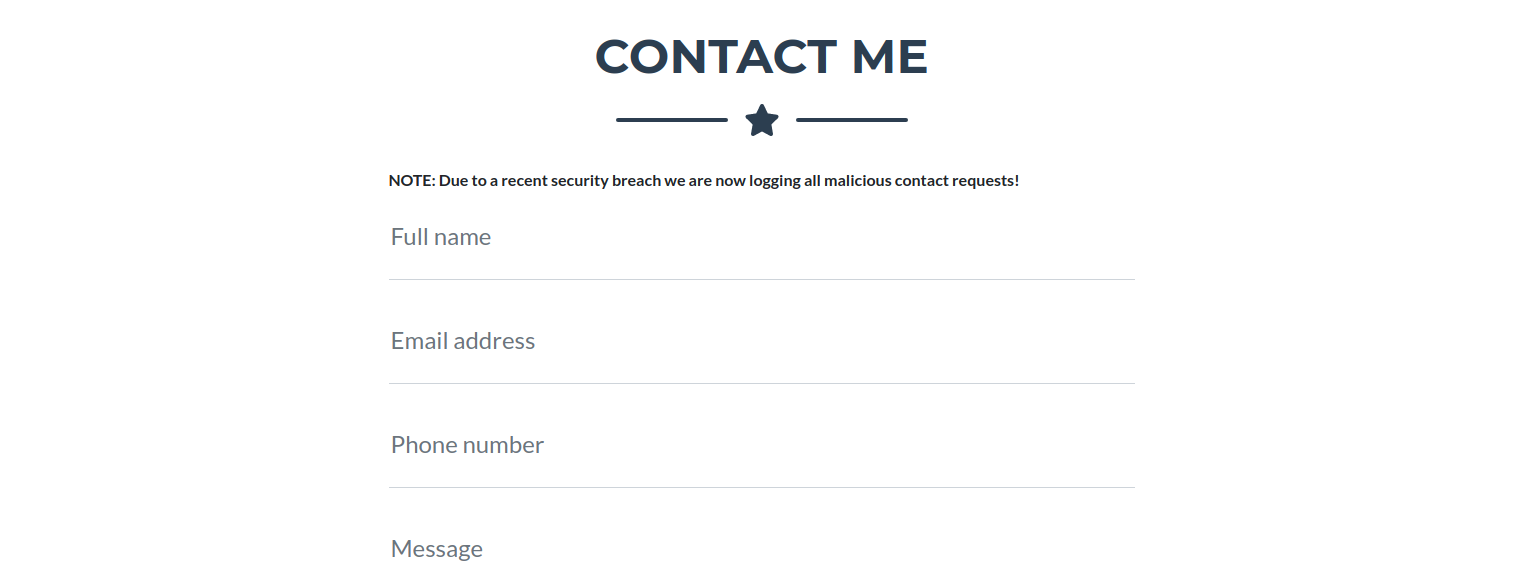
# Log Injection

Web applications often log operational details to log files. This includes request details such as source IP address, request path, and request parameters. This is done to simplify debugging in case of errors as well as allow for log analysis in case of security incidents. Furthermore, a web application may implement additional security measures such as Web Application Firewalls (WAFs) which log additional data that is deemed suspicious, for instance, if a request contains certain special characters that may indicate an exploitation attempt of an injection vulnerability. If user input is written into log files without sanitization of CRLF characters, it might be possible to forge log entries in a [Log Injection](https://owasp.org/www-community/attacks/Log_Injection) attack or even escalate to Cross-Site Scripting or Remote Code Execution via log poisoning.

## Identification

The exercise below is a simple web application that implements a contact form:

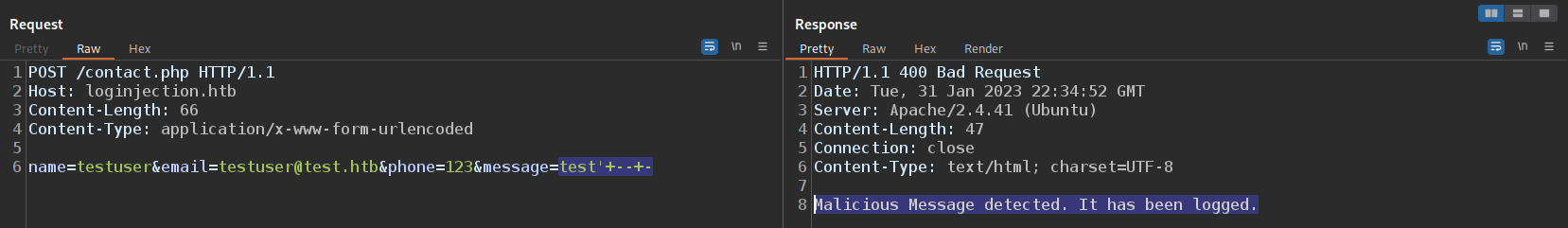


The security notice tells us that the web application implements custom logging of malicious requests, similar to a WAF. Playing with the contact form and submitting different messages indicates that certain special characters seem to be blocked by a blacklist filter. Here is an example of a blocked request containing a potential SQL injection payload:

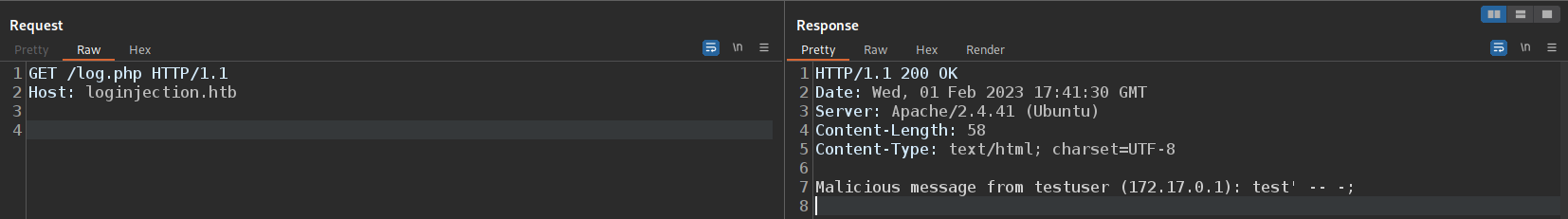
Code: http

POST /contact.php HTTP/1.1  
Host: loginjection.htb  
Content-Length: 66  
Content-Type: application/x-www-form-urlencoded  
  
[name=testuser&email=testuser@test.htb&phone=123&message=test'+--+-](mailto:name=testuser&email=testuser@test.htb&phone=123&message=test'+--+-)

This request results in the following behavior:



For demonstration purposes, the web application implements an additional endpoint at /log.php that displays the log file. While this might seem unrealistic, many web applications implement such a functionality. However, it is typically hidden behind authentication such that only admin users are allowed to access it. In some rare cases, the web application might incorrectly store the log file in the current working directory making it publicly accessible. Thus it might be a good idea to fuzz for files with a .log extension on the web server. We can see the format messages are logged in when accessing /log.php:

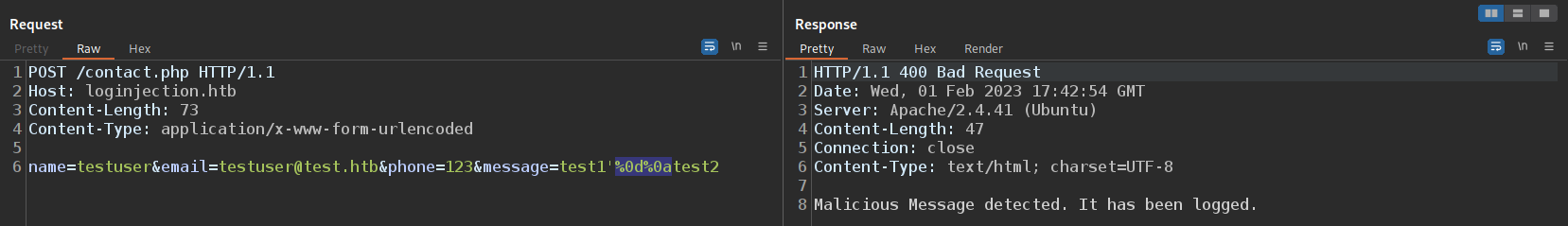


The log files contain our IP address as well as the provided username and message. Special characters are not encoded. We can test whether the CRLF sequence is properly sanitized by including the URL-encoded sequence %0d%0a in our message:

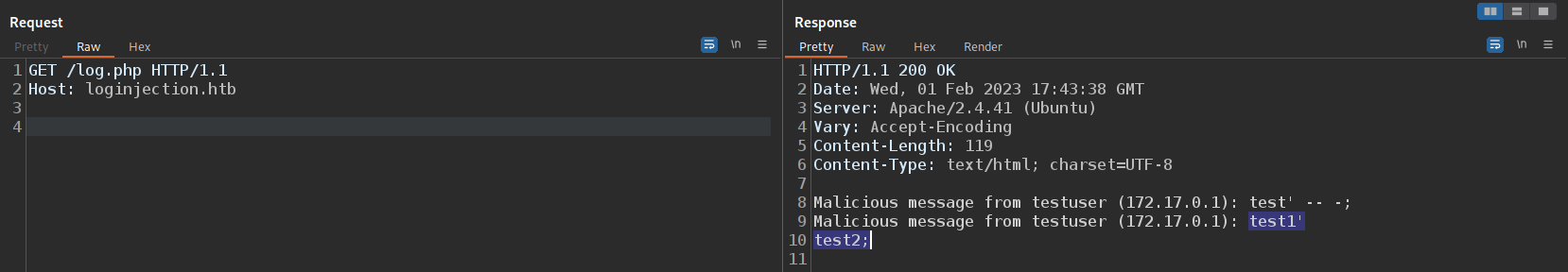
Code: http

POST /contact.php HTTP/1.1  
Host: loginjection.htb  
Content-Length: 73  
Content-Type: application/x-www-form-urlencoded  
  
[name=testuser&email=testuser@test.htb&phone=123&message=test1'%0d%0atest2](mailto:name=testuser&email=testuser@test.htb&phone=123&message=test1'%0d%0atest2)

This message is also logged:



We can confirm that we successfully injected a newline into the log file:



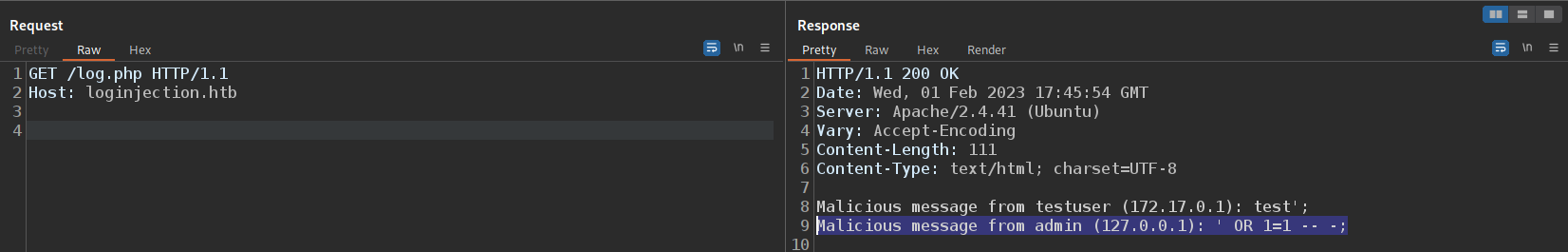
## Exploitation

A classical log injection vulnerability like the example discussed above could be exploited by forging a log entry to make it seem like another user took a malicious action. In our example, this can be done by sending a request similar to the following:

Code: http

POST /contact.php HTTP/1.1  
Host: 172.17.0.2  
Content-Length: 124  
Content-Type: application/x-www-form-urlencoded  
  
name=testuser&email=testuser%40test.htb&phone=123&message=test1';%0a%0dMalicious+message+from+admin+(127.0.0.1):+'+OR1=1+--+-

This request injects an additional line into the log file that makes it seem like the admin user tried to exploit a SQL injection. We can confirm this by looking at the log file:



We can see that we successfully injected a forged log entry. This effectively invalidates the log file when the vulnerability is discovered, as the system administrators cannot be sure which log entries are real and which ones are forged.

#### Log Poisoning

Log files can also be used to achieve remote code execution if PHP code can be injected. This also works in our lab, however, in a real-world setting we would typically need to exploit a Local File Inclusion (LFI) vulnerability first to obtain RCE via log poisoning. For more details on LFIs, check out the [File Inclusion](https://academy.hackthebox.com/module/details/23) module. We are not discussing log poisoning in more detail here, however, we can obtain remote code execution by injecting PHP code with a request like this:

Code: http

POST /contact.php HTTP/1.1  
Host: 172.17.0.2  
Content-Length: 80  
Content-Type: application/x-www-form-urlencoded  
  
name=testuser&email=testuser%40test.htb&phone=123&message=<?php+echo+'pwned';+?>

In a real-world setting, filters may be in place that we need to bypass.