



---

# WIRESHARK

---

SAD



2ºASIR  
I.E.S. ANTONIO MACHADO  
ANA OROZCO ASENSIO

## Contenido

Introducción.....	2
Monitoreo.....	2
MySQL .....	2
HTTP.....	3
Conclusión.....	5

# Introducción.

En esta práctica vamos a usar Wireshark para monitorear la red usando Wordpress con MySQL y Apache con PHP.

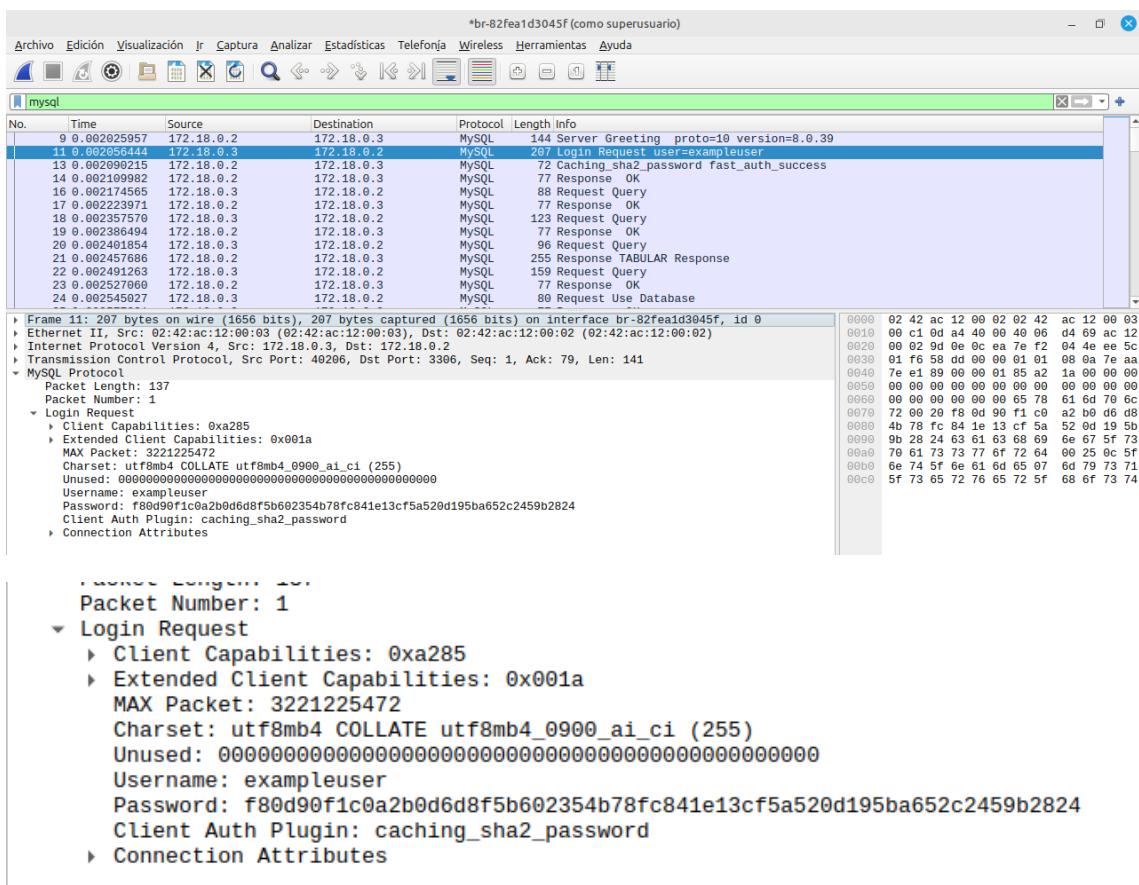
Comprobaremos lo que podemos ver al buscar en los paquetes que monitoreamos y buscaremos la contraseña y usuario de Wordpress.

# Monitoreo.

Voy a dividir en 2 este monitoreo.

MySQL

En esta captura podemos ver el tráfico de la red filtrando con el protocolo MySQL, al entrar en los login request podemos ver que la contraseña está cifrada, por lo que de estos paquetes podemos sacar poca información.



## HTTP.

Con el protocolo HTTP vemos varios tipos de paquetes como GET que solicita los recursos al servidor o POST que envía datos al servidor (se suele usar para los formularios)

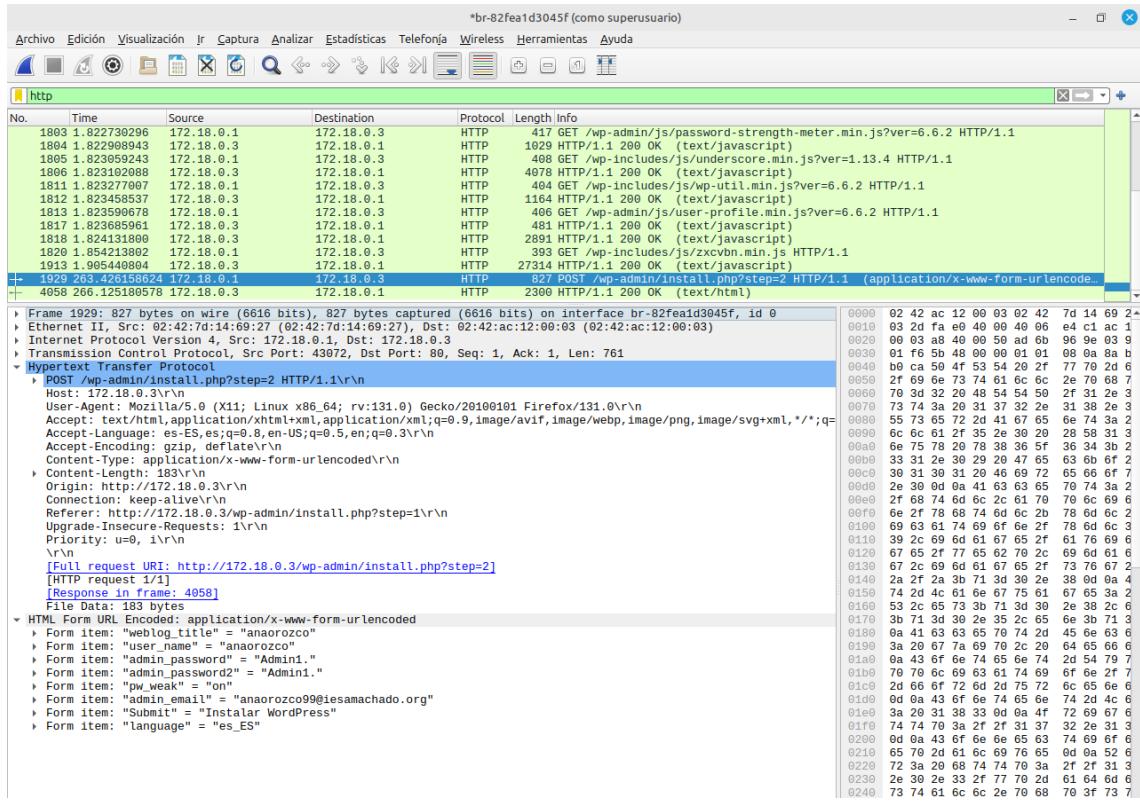
The screenshot shows a Wireshark capture of network traffic. The top pane displays a list of frames, mostly GET requests for wp-admin/install.php, with one POST request at the bottom. The bottom pane shows the details for frame 4058, which is a POST request for wp-admin/install.php?step=2. The hex and ASCII panes show the raw data and readable text respectively, including the JSON payload of the POST request.

No.	Time	Source	Destination	Protocol	Length	Info
4	9.0000753895	172.18.0.4	172.18.0.3	HTTP	500	POST /wp-admin/install.php?step=1 HTTP/1.1 (application/x-www-form-urlencoded)
1793	1.8219058801	172.18.0.3	172.18.0.1	HTTP	3446	HTTP/1.1 500 OK (text/html)
1792	1.8219058110	172.18.0.1	172.18.0.3	HTTP	497	GET /wp-includes/js/wp-async.min.js?ver=1.0 HTTP/1.1
1794	1.822175217	172.18.0.3	172.18.0.1	HTTP	664	HTTP/1.1 200 OK (text/javascript)
1795	1.822308279	172.18.0.1	172.18.0.3	HTTP	422	GET /wp-includes/js/dist/hooks.min.js?ver=2810c76e705dd1a53b18 HTTP/1.1
1796	1.822486417	172.18.0.1	172.18.0.3	HTTP	421	GET /wp-includes/js/dist/118n.min.js?ver=5e580eb46a90c2b997e6 HTTP/1.1
1800	1.822507575	172.18.0.3	172.18.0.1	HTTP	1951	HTTP/1.1 200 OK (text/javascript)
1803	1.822730294	172.18.0.1	172.18.0.3	HTTP	417	GET /wp-admin/js/password-strength-meter.min.js?ver=0.6.2 HTTP/1.1
1804	1.822998943	172.18.0.3	172.18.0.1	HTTP	1029	HTTP/1.1 200 OK (text/javascript)
1805	1.823059243	172.18.0.1	172.18.0.3	HTTP	408	GET /wp-includes/js/underscore.min.js?ver=1.13.4 HTTP/1.1
1806	1.823102088	172.18.0.3	172.18.0.1	HTTP	4078	HTTP/1.1 200 OK (text/javascript)
1811	1.823277007	172.18.0.1	172.18.0.3	HTTP	404	GET /wp-includes/js/wp-util.min.js?ver=6.6.2 HTTP/1.1
1812	1.823458537	172.18.0.3	172.18.0.1	HTTP	1164	HTTP/1.1 200 OK (text/javascript)
1813	1.823590678	172.18.0.1	172.18.0.3	HTTP	406	GET /wp-admin/js/user-profile.min.js?ver=6.6.2 HTTP/1.1
1817	1.823685961	172.18.0.3	172.18.0.1	HTTP	481	HTTP/1.1 200 OK (text/javascript)
1818	1.824131808	172.18.0.3	172.18.0.1	HTTP	2891	HTTP/1.1 200 OK (text/javascript)
1822	1.854213802	172.18.0.1	172.18.0.3	HTTP	393	GET /wp-includes/js/zxcvbn.min.js HTTP/1.1
1913	1.965440884	172.18.0.3	172.18.0.1	HTTP	27314	HTTP/1.1 200 OK (text/javascript)
1929	263.426158624	172.18.0.1	172.18.0.3	HTTP	827	POST /wp-admin/install.php?step=2 HTTP/1.1 (application/x-www-form-urlencoded)
4058	266.125180578	172.18.0.3	172.18.0.1	HTTP	2300	HTTP/1.1 200 OK (text/html)

En esta captura vemos que solo tenemos códigos del estado del HTTP como 200 OK que significa que la solicitud se ha procesado correctamente, también tenemos el 500 Internal Server Error que dice que ha ocurrido un error en el servidor y el famoso 404 Not Found, que lo que nos dice es que no se encuentra el recurso que hemos solicitado. Además de estos hay más errores pero esto son los más comunes.

Seguimos buscando podemos encontrar información como el navegador que se utiliza en el tipo POST, el sistema operativo e incluso el idioma.

En los paquetes GET suele estar la información de la versión de HTTP utilizada, la url del recurso, la versión del PHP, el tipo de contenido, etc.



Además, podemos ver el usuario y la contraseña usados en el formulario de creación, por lo que no hay cifrado y por lo tanto no es seguro.

```

▼ HTML FORM URL Encoded: application/x-www-form-urlencoded
  ▼ Form item: "weblog_title" = "anaorozco"
    Key: weblog_title
    Value: anaorozco
  ▼ Form item: "user_name" = "anaorozco"
    Key: user_name
    Value: anaorozco
  ▼ Form item: "admin_password" = "Admin1."
    Key: admin_password
    Value: Admin1.
  ▼ Form item: "admin_password2" = "Admin1."
    Key: admin_password2
    Value: Admin1.
  ▶ Form item: "pw_weak" = "on"
  ▼ Form item: "admin_email" = "anaorozco99@iesamachado.org"
    Key: admin_email
    Value: anaorozco99@iesamachado.org
  ▶ Form item: "Submit" = "Instalar WordPress"
  ▶ Form item: "language" = "es_ES"

```

## Conclusión

Después de esta práctica me he dado cuenta de por qué no es seguro el protocolo HTTP y deberíamos usar el HTTPS (aunque no todo el contenido de HTTPS está cifrado).

La diferencia que veríamos en wireshark sería que al hacer el monitoreo veríamos los paquetes, pero estos tendrían la información encriptada y autenticada