

REPORT TECNICO: VULNERABILITY ASSESSMENT & EXPLOITATION (DVWA)

Data: 15 Gennaio 20267

Autore: Gerald Bejte

Corso: Cybersecurity Epicode

Target: Damn Vulnerable Web Application (DVWA) v1.0.7

1. INTRODUZIONE TEORICA

1.1 Cross-Site Scripting (XSS) Reflected

L'XSS è una vulnerabilità che permette a un attaccante di iniettare script malevoli (solitamente JavaScript) nelle pagine web visualizzate da altri utenti. Nella variante **Reflected**, lo script viene "riflettuto" dal server web alla vittima tramite un parametro (come una query di ricerca) che non viene correttamente filtrato.

In parole semplici: è come incollare un ordine malevolo su un foglio che il server rilegge ad alta voce al browser, costringendolo a eseguirlo.

1.2 SQL Injection (SQLi)

La SQL Injection è una tecnica che sfrutta la mancata validazione degli input dell'utente per manipolare le query inviate al database. Inserendo caratteri speciali (come l'apice '), l'attaccante può interrompere la logica della query originale e inserire comandi arbitrari.

In parole semplici: si usa la "grammatica" del database per convincerlo a rivelare segreti o bypassare i controlli di sicurezza.

2. CONFIGURAZIONE DEL LABORATORIO

2.1 Verifica Connattività

L'ambiente è composto da una macchina attaccante (**Kali Linux**) e una vittima (**Metasploitable/DVWA**). La comunicazione è stata verificata tramite il protocollo ICMP.

- **IP Attaccante:** [Tuo IP Kali]
- **IP Target:** 192.168.65.20 / 192.168.50.102

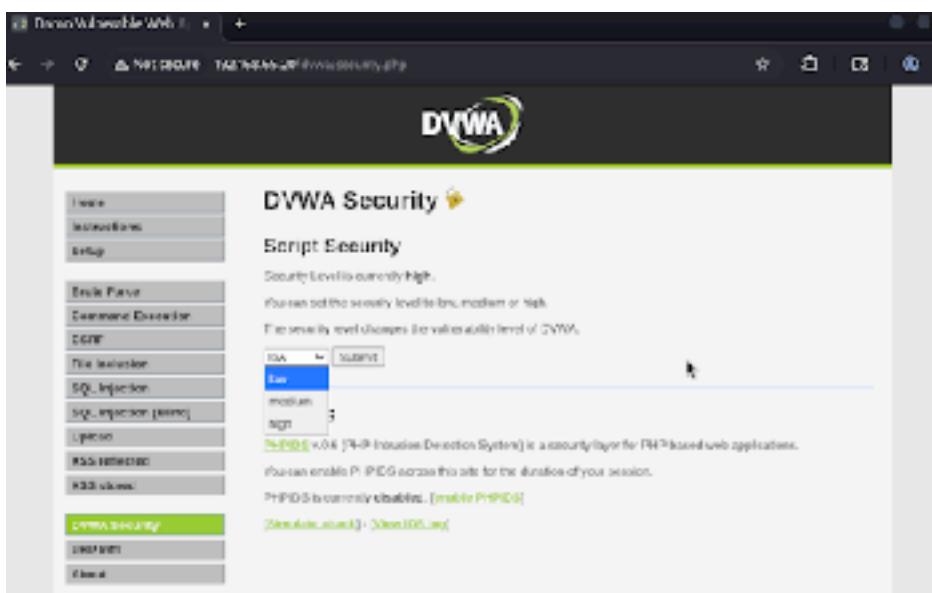
```
valid till forever preferred till forever

└─(kali㉿kali)-[~]
$ ping 192.168.65.20
PING 192.168.65.20 (192.168.65.20) 56(84) bytes of data.
64 bytes from 192.168.65.20: icmp_seq=1 ttl=64 time=3.45 ms
64 bytes from 192.168.65.20: icmp_seq=2 ttl=64 time=1.89 ms
64 bytes from 192.168.65.20: icmp_seq=3 ttl=64 time=1.73 ms
64 bytes from 192.168.65.20: icmp_seq=4 ttl=64 time=1.68 ms
64 bytes from 192.168.65.20: icmp_seq=5 ttl=64 time=1.86 ms
64 bytes from 192.168.65.20: icmp_seq=6 ttl=64 time=2.07 ms
64 bytes from 192.168.65.20: icmp_seq=7 ttl=64 time=1.87 ms
64 bytes from 192.168.65.20: icmp_seq=8 ttl=64 time=1.53 ms
64 bytes from 192.168.65.20: icmp_seq=9 ttl=64 time=1.15 ms
64 bytes from 192.168.65.20: icmp_seq=10 ttl=64 time=1.44 ms
^C
--- 192.168.65.20 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9034ms
rtt min/avg/max/mdev = 1.005/1.098/3.452/0.617 ms
```

```
imc6 ::1/128 scope host  
    valid_lft forever preferred_lft forever  
eth0: <NORAIDCAST,BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000  
    linklayer brd ::12:00:02:b2:20 brd ::ffff:ffff:ffff:ffff:ffff  
    inet 192.168.65.20/24 brd 192.168.65.255 scope global eth0  
        inet6 fe80::3122:3bff:fe62:b220%64 scope link dynamic  
            valid_lft 2591990sec preferred_lft 604790sec  
    imc6 fe80::3e12:d0ff:fe62:b220%64 scope link  
        valid_lft forever preferred_lft forever  
#admin@metasploitable:~$  
#admin@metasploitable:~$ ping 192.168.65.10  
PING 192.168.65.10 (192.168.65.10) 56(44) bytes of data.  
+ bytes from 192.168.65.10: icmp_seq=1 ttl=64 time=4.47 ms  
+ bytes from 192.168.65.10: icmp_seq=2 ttl=64 time=1.17 ms  
+ bytes from 192.168.65.10: icmp_seq=3 ttl=64 time=0.874 ms  
bytes from 192.168.65.10: icmp_seq=4 ttl=64 time=1.26 ms  
bytes from 192.168.65.10: icmp_seq=5 ttl=64 time=0.817 ms  
bytes from 192.168.65.10: icmp_seq=6 ttl=64 time=1.12 ms  
bytes from 192.168.65.10: icmp_seq=7 ttl=64 time=1.68 ms  
bytes from 192.168.65.10: icmp_seq=8 ttl=64 time=1.34 ms  
192.168.65.10 ping statistics
```

2.2Impostazione Target

Per procedere con l'esercitazione, il livello di sicurezza della DVWA è stato impostato su **LOW**. Questa modalità disabilita i filtri di input lato server, permettendo l'esecuzione di payload base.



3. EXPLOITATION: XSS REFLECTED

3.1 Identificazione della vulnerabilità

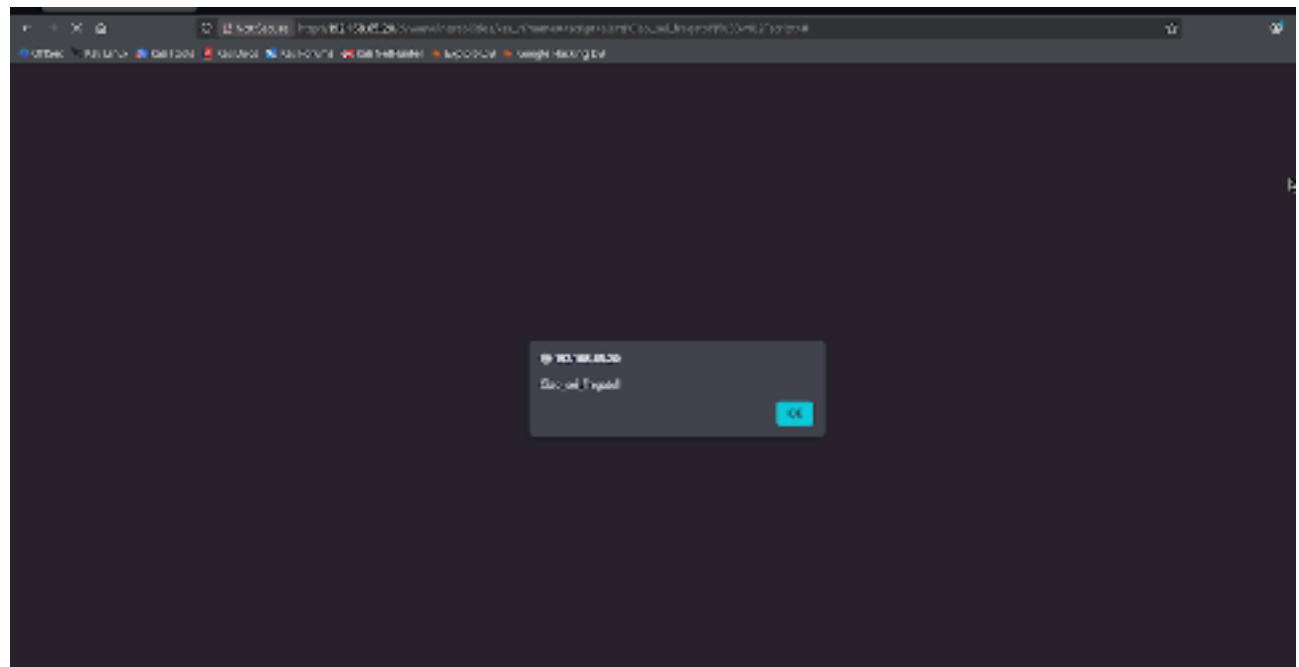
Navigando nella sezione "XSS reflected", è stato individuato un campo di input, l'applicazione lo riflette direttamente a video.

3.2 Esecuzione del Payload

Per confermare la vulnerabilità, è stato iniettato il seguente script:

HTML

```
<script>alert('Ciao_sei_fregato!');</script>
```



Esito: Il browser ha interpretato i tag HTML ed eseguito il codice JavaScript, mostrando un pop-up di avviso.

4. EXPLOITATION: SQL INJECTION

L'obiettivo è manipolare la query originale per forzare il database a rivelare informazioni che non dovrebbero essere accessibili.

4.1 Identificazione della vulnerabilità

Nella sezione "SQL Injection", l'applicazione richiede un User ID per restituire i dati dell'utente.

Per estrarre tutti i record presenti nella tabella degli utenti senza conoscere gli ID, è stata sfruttata la logica booleana inserendo il seguente payload nel campo ID:

SQL

' OR 1=1 --

Analisi del Payload:

- ': Chiude la stringa di ricerca originale.
- ' OR 1=1: Introduce una condizione sempre vera, forzando il database a restituire ogni riga.
- : Commenta il resto della query originale per evitare errori di sintassi.

The screenshot shows the DVWA SQL Injection page. On the left, there's a sidebar with various exploit categories: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (the current page), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, and About. The main area has a title 'Vulnerability: SQL Injection'. It contains a form with 'User ID:' and a 'Submit' button. Below the form, the output shows five user records extracted from the database:

ID	First name	Surname
' OR '1='1 --	admin	admin
' OR '1='1 --	Gordon	Brown
' OR '1='1 --	Hack	Mk
' OR '1='1 --	Pablo	Picasso
' OR '1='1 --	Bob	Smith

At the bottom, there's a 'More info' link.

Esito: Il database ha restituito l'elenco completo di nomi e cognomi degli utenti registrati nel sistema.

Estrazione Utenti e Password:

Oltre al classico bypass di autenticazione, è possibile utilizzare l'operatore UNION per unire i risultati della query legittima con quelli di una tabella a nostra scelta.

- **Payload Avanzato:** ' UNION SELECT user, password FROM users #

Analisi del comando:

- ': Chiude il campo ID previsto dal programmatore.
- **UNION SELECT user, password:** Ordina al database di aggiungere alla risposta le colonne user e password.
- **FROM users:** Specifica la tabella da cui estrarre i dati riservati.
- #: Commenta e annulla il resto della query originale.

Vulnerability: SQL Injection

User ID: Submit

SQL syntax error near ' or '' at position 1

ID = '' UNION SELECT user, password FROM users #
First name: admin
Surname: 5f44c2393ea769c61d8327de1862cf99

ID = '' UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c429cb39c5f268853518922e00

ID = '' UNION SELECT user, password FROM users #
First name: 1337
Surname: 9d5123d75a+3c36647e6d4fc692108

ID = '' UNION SELECT user, password FROM users #
First name: pablo
Surname: 6d183d9ff5bbe40c4de36e5c11e9e9b7

ID = '' UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f44cc950aa76ac61a85f4d80882cf99

More info

Risultati dell'estrazione: L'attacco ha permesso di ottenere non solo gli hash delle password, ma l'intero elenco degli **Username** presenti nel database.

- **Dati Compromessi:** Elenco completo degli utenti (es. **admin**, **gordonb**, **1337**, **pablo**, **smithy**).
- **Impatto:** La compromissione dei nomi utente facilita enormemente le fasi successive dell'attacco. Conoscendo lo username, un malintenzionato può avviare attacchi di **Password Spraying** o utilizzare i nomi utente in combinazione con i rispettivi hash MD5 per attacchi offline.

Nota tecnica sugli Hash MD5: Le password sono state estratte sotto forma di hash MD5. Data l'obsolescenza dell'algoritmo e l'assenza di un "salt" (un valore casuale aggiunto per rendere l'hash

unico), è possibile risalire al testo in chiaro in pochi secondi tramite attacchi di tipo **Dictionary** o consultando **Rainbow Tables** online.

5. CONCLUSIONI E RACCOMANDAZIONI

L'analisi dimostra che l'assenza di **sanitizzazione dei caratteri speciali** (come < > ' #) e la mancanza di controlli sugli input degli utenti rendono l'infrastruttura estremamente vulnerabile ad attacchi che possono compromettere l'integrità dei dati e la sessione degli utenti.

Raccomandazioni Tecniche:

1. **Utilizzo di Prepared Statements (Query Parametrizzate):** Per la difesa da SQLi, è fondamentale separare il codice SQL dai dati inseriti dall'utente, impedendo che questi vengano interpretati come comandi.
2. **Output Encoding:** Per prevenire l'XSS, ogni dato riflesso sulla pagina deve essere convertito in entità HTML (es. < diventa < ;), neutralizzando l'esecuzione di script.
3. **Principio del Minimo Privilegio:** L'utente del database utilizzato dall'applicazione web non dovrebbe avere i permessi per accedere a tabella sensibili o eseguire comandi amministrativi.

Le vulnerabilità riscontrate evidenziano l'importanza di:

1. **Input Validation:** Non fidarsi mai dell'input dell'utente.
2. **Output Encoding:** Convertire i caratteri speciali HTML (es: < in < ;) per prevenire XSS.
3. **Prepared Statements:** Utilizzare query parametrizzate per impedire che l'input dell'utente venga interpretato come codice SQL.

