Zeste de savoir

La vulnérabilité Shellshock

12 août 2019

Table des matières

Après *Heartbleed* , qui permettait d'attaquer OpenSSL pour récupérer la clef privée utilisée par un serveur vulnérable[1], c'est au tour de *Shellshock* de faire parler d'elle. Cette fois-ci, c'est Bash qui est touché et une exploitation de cette faille permet d'exécuter assez facilement du code arbitraire. De quoi laisser songeur...

En réalité, maintenant que les gens savent où regarder, ce n'est pas une, mais au moins six failles qui ont été découvertes. Cet article ne parle que de celle par qui « tout a commencé ».
Quelques CVE à regarder pour les intéressés :

- 2014-6271
- 2014-6277
- 2014-6278
- 2014-7169
- 2014-7186
- 2014-7187

1. Le coupable : Bash

Bash est le *shell* du projet GNU publié sous licence libre (GPL3 pour les versions plus récentes[3]). La première version de Bash date de 1988, ce qui en fait un projet vénérable. C'est une donnée intéressante, car il paraitrait que Shellshock est exploitable dans Bash depuis une vingtaine d'années[2]. C'est une donnée intéressante à prendre en compte et très symbolique, car l'un des arguments en faveur du logiciel libre est la prétendue facilité de faire des audits « soi-même ». Quiconque a déjà regardé le code de Bash se rendra vite compte que les choses ne sont pas si simples. De la même manière, *Heartbleed* avait mis plus de deux ans à être découverte, alors qu'OpenSSL est une pièce logicielle majeure de l'internet de confiance d'aujourd'hui. À ce sujet, la FSF s'est fendu d'un article à ce sujet . Sa lecture alimente le débat.

1. Et c'est même plus facile que ça en a l'air. Les gars de chez CloudFlare ont écrit un article très intéressant à ce sujet et ce ne sont bien évidemment pas les seuls .
2. pas prouvé
3. Apple utilise un *fork* interne de Bash, basé sur une version antérieure publiée sous GPL2 qu'ils font évoluer en parallèle.
4. Notez l'usage du conditionnel.

2. La vulnérabilité

Bash offre plusieurs fonctionnalités peu connues. L'une d'entre elles consiste à définir une variable d'environnement pour une commande donnée.

```
:::console
sh$ env TEST="Bonjour" bash -c "printenv TEST"
Bonjour
sh$ printenv TEST
# error
```

La vulnérabilité Shellshock profite d'une erreur des développeurs de Bash justement dans cette fonctionnalité. Plus précisément, le loup se cache dans un cas particulier comme celui-ci :

```
:::console
sh$ env TEST='() { echo test; };' bash -c TEST
test
```

Ici, la variable d'environnement TEST contient une fonction qui exécutée par l'appel de `bash`. À noter que la fonction n'est pas exécutée si elle n'est pas appelée. Ça peut sembler légitime, mais ça aura très vite son importance. Ainsi :

```
:::console
sh$ env TEST='() { echo test; };' bash -c "echo tada"
tada
```

Il est donc possible de fournir du code arbitraire à une commande Bash *via* une variable d'environnement. Ce code ne sera exécuté que sur la demande de la commande. À noter que si ladite commande s'attend à ce que la variable d'environnement ne contienne qu'une chaine de caractère et l'utilise comme telle, le code ne sera pas non plus exécuté :

```
:::console
sh$ env TEST='() { echo test; };' bash -c "printenv TEST"
() { echo test
}
```

Ce que les gens de Bash n'ont certainement pas voulu implémenter, cependant, c'est ce comportement en particulier :

```
:::console
sh$ env TEST='() { echo test; }; echo "vulnerable"' bash -c "echo
    hello"
vulnerable
hello
```

Qu'est-ce qui est choquant ici ? Il faut regarder d'un peu plus près la commande qui, il faut l'avouer, n'est pas forcément très familière ou lisible. Globalement, elle est séparée en deux parties : `env TEST='() { echo test; }; echo "vulnerable"'` et `bash -c "echo hello"`. Nous avons déjà vu la sémantique de tout cela : cela rajoute une variable d'environnement `TEST` pour l'exécution de la commande `bash -c "echo hello"`. Cette variable d'environnement est définie comme une fonction qui affiche « test »... suivie d'une instruction `echo "vulnerable"`.

Quand on regarde le résultat, on voit que la commande `echo hello` a bien été exécutée. On voit aussi que la fonction stockée dans `TEST`, elle, ne l'a pas été. Par contre, et c'est là que le bât blesse, l'instruction l'a été. On voit que « vulnerable » a été imprimée à l'écran. Shellshock, c'est cette exécution involontaire.

3. Pourquoi c'est mal ? Un exemple avec Apache et les scripts CGI

Il n'est pas forcément évident de voir quel est le problème ici. En tout cas, je ne l'ai personnellement pas vu directement. Ça devient tout de suite plus clair quand on s'intéresse à l'utilisation qui est faite de la fonctionnalité à l'origine du scandale. Le fonctionnement d'Apache et des scripts CGI est un exemple à la fois parlant et simple à comprendre.

CGI stocke les headers de vos requêtes HTTP dans des variables d'environnement. Et si vous vous amusez à tricher avec votre User Agent ?

```
:::console
curl -k http://site-vulnerable.org/cgi-bin/test -H "User-Agent: ()
    { :;}; cp /etc/passwd /srv/http/public"
```

L'argument `-H` de `curl` permet de redéfinir le header de votre requête HTTP. Ce qui veut dire que votre User Agent aura comme valeur, lorsque `curl` fera une requête HTTP pour télécharger la page demandée, `User-Agent: () { :;}; cp /etc/passwd /srv/http/public`.

Conséquence ? Une variable d'environnement pour l'User-Agent sera crée... Exploitant la vulnérabilité Shellshock, l'instruction `cp /etc/passwd /srv/http/public` sera exécutée !

4. Contremesures

Pour savoir si votre machine utilisant Bash (Linux ou OS X, par exemple) est vulnérable, vous pouvez simplement utiliser la commande présentée à la fin de la Section « Vulnérabilité », à savoir :

```
:::console
sh$ env TEST='() { echo test; }; echo "vulnerable"' bash -c "echo
    hello"
```

Si « vulnerable » apparait sur votre écran, c'est que vous êtes vulnérables. Dans ce cas-là, il vous faudra patcher/mettre à jour votre version de bash. À noter que les *patchs* qui ont été émis ne sont que des correctifs temporaires. La vraie solution pour contrer Shellshock est de recoder un parseur beaucoup plus robuste. Le vrai danger de Shellshock réside dans son extrême facilité d'exploitation et sa présence dans une très large majorité des serveurs. Il ne sera pas si évident de s'en débarrasser, notamment dans les systèmes embarqués. Affaire à suivre, donc.

> Crédit icône : [http ://aaronkondziela.com/2014/09/shellshock-logo-for-bash-vulnerability/](Aaron K.) (CC BY 4.0)

Liste des abréviations

Bash Bourne-again Shell. 1, 2, 4

CVE Common Vulnerabilities and Exposures. 1

FSF Free Software Foundation. 1