TP 1 : Environnement cloisonné (correction)

Romain CARRÉ
romain.carre@cea.fr

1 Environnement de travail

– ⚠ La quasi-totalité des actions à réaliser nécessitent les droits de super-utilisateur.

```
sudo su
```

– installez `thttpd` et `wget` à l'aide de votre gestionnaire de paquets habituel.

```
apt-get install thttpd wget
```

– récupérez les différents fichiers du TP sur la clef USB de votre encadrant et placez-les dans un répertoire au choix que l'on désignera ici par *wwwroot*.

```
export wwwroot=/tmp
cp -r /media/usb/* $wwwroot
```

2 Utilisation sans protection

– lancez le *daemon* `thttpd` depuis le répertoire *wwwroot* avec la commande :

```
cd $wwwroot
thttpd -D -u root -c '**.sh' -nor
```

– à quoi servent les différentes options présentes sur la ligne de commande ?

`-D`	: lance le *daemon* au premier plan (*foreground*)
`-u root`	: les processus fils sont lancés au nom de root
`-c '**.sh'`	: autorise l'extension sh à contenir des scripts
`-nor`	: n'active pas la *chroot jail* applicative par défaut

– en quoi reflètent-elles correctement le titre de cette partie du TP ? expliquez.

Au niveau de son système de fichier, le *daemon* n'est confiné :
– ni par une *chroot jail* manuelle ;
– ni par une *chroot jail* applicative.
Il n'est clairement pas confiné **du tout**.

– ouvrez un navigateur web, et chargez localement la page SH téléchargée.

FIGURE 1 – Aucune protection - Page vulnérable

– testez-la, par exemple en la faisant s'ouvrir elle-même. une idée de la faille ?

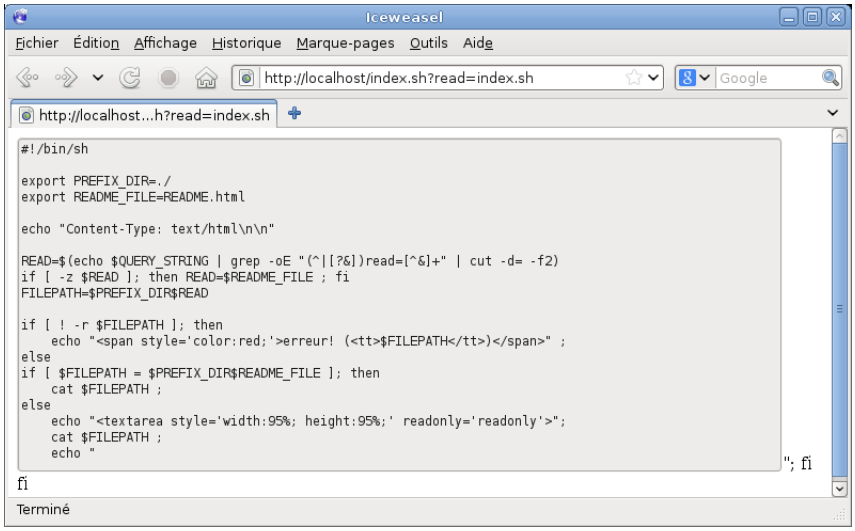

FIGURE 2 – Aucune protection - La page s'inclut elle-même

La vulnérabilité est probablement un *directory traversal*. Elle consiste à abuser de la non-vérification du chemin d'accès demandé au serveur web pour consulter des fichiers qu'il n'était pas prévu de fournir à l'utilisateur au départ. En pratique, on remonte dans les répertoires en les « traversant » (d'où son nom), pour avoir accès à la racine du système de fichiers. Dès lors, en « redescendant », on peut lire n'importe quel fichier accessible par l'utilisateur avec lequel est lancé le serveur. (consulter `http://en.wikipedia.org/wiki/Directory_traversal_attack`)

– parcourez son code source, mettez en évidence la vulnérabilité, exploitez-là.

```
READ=$(echo $QUERY_STRING | grep -oE "(^|[?&])read=[^&]+" | cut -d= -f2)
FILEPATH=$PREFIX_DIR$READ                    # <-- la vulnérabilité est ici !
```

Pour l'exploiter, c'est relativement simple : on demande l'accès à un fichier depuis la racine du système, en ajoutant successivement le préfixe `../`. Par exemple, si on veut connaître la version courante du système, le fichier `/etc/debian_version`.

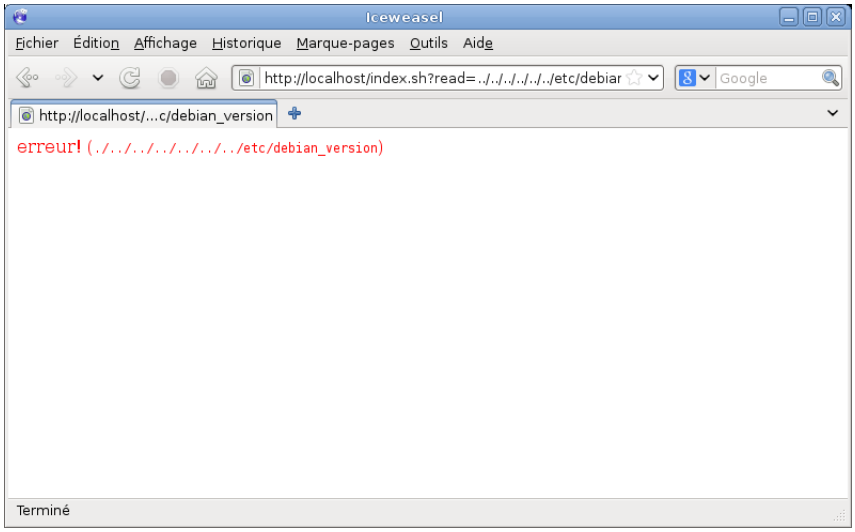

FIGURE 3 – Aucune protection - ../../../../../../etc/debian_version

– et avec un `../` de plus :

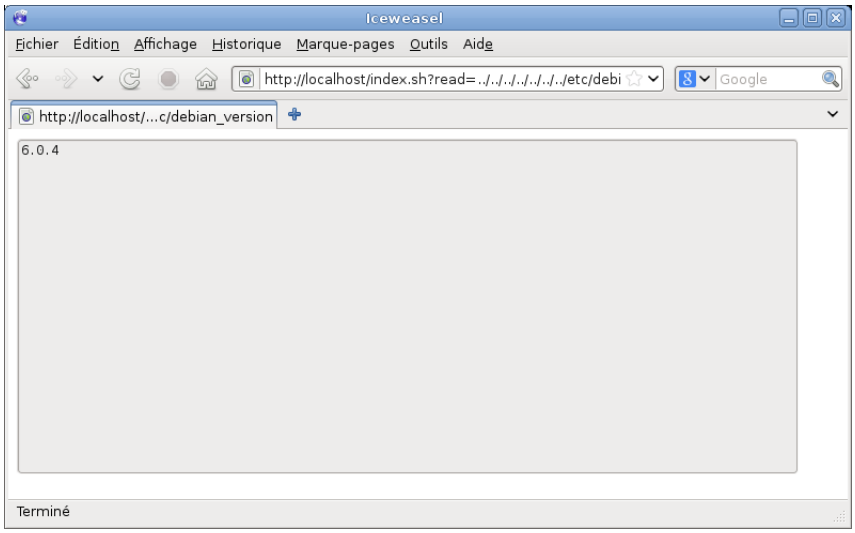

FIGURE 4 – Aucune protection - ../../../../../../../etc/debian_version

– en tant qu'attaquant, quels fichiers intéressants consulteriez-vous grâce à elle ?
Il peut être intéressant, si le serveur web est lancé en tant que root, de consulter les fichiers `/etc/passwd` ou `/etc/shadow`, qui contiennent respectivement les logins des comptes utilisateurs locaux, ou les hashés de leur mot de passe. On pourrait imaginer aussi vouloir récupérer la configuration réseau, celle du pare-feu, les mots de passes administrateurs d'autres serveurs spécifiques, etc.
– en tant que professionnel de sécurité, que feriez-vous pour y remédier ?
Pour remédier à **ce problème particulier**, on peut mettre en place une *chroot jail*, c'est à dire une protection par confinement. C'est d'ailleurs le sujet du TP !

3 Construction d'une cage

– construisez une *chroot jail* autour du *daemon* `thttpd`

```
# création de l'arborescence de la cage  :
mkdir -p bin dev etc lib lib64 var/log var/run var/www

# copie de tous les fichiers nécessaires  :
cp $(which thttpd) $(which sh) $(which cut) $(which grep) $(which cat) bin/
cp /etc/passwd /etc/nsswitch.conf etc/
ldd bin/*
    bin/thttpd :
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x00007f61294d9000)
        libc.so.6 => /lib/libc.so.6 (0x00007f6129177000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f612972c000)
    bin/grep :
        libdl.so.2 => /lib/libdl.so.2 (0x00007fee09eeb000)
        libc.so.6 => /lib/libc.so.6 (0x00007fee09b89000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fee0a10b000)
cp /lib/libc.so.6 /lib/libcrypt.so.1 /lib/libdl.so.2 lib/
cp /lib64/ld-linux-x86-64.so.2                       lib64/
cp $wwwroot/*                                        var/www/

# configuration du daemon rsyslogd  :
cat << EOF > /etc/rsyslog.d/thttpd.conf
$AddUnixListenSocket            /chrootjail/dev/log
:programname, isequal, "thttpd" /chrootjail/var/log/thttpd.log
EOF
/etc/init.d/rsyslogd restart
```

– relancez le *daemon* en vous plaçant à la racine de votre *chroot jail*.

```
chroot . bin/thttpd -D -u root -c '**.sh' -nor -d var/www -i /var/run/thttpd.pid
```

– en quoi cette nouvelle commande reflète le titre de cette partie du TP ?
La commande ici est en réalité une commande `chroot` : on ne lance pas `thttpd` directement. Cela correspond donc à construire une *chroot jail* et à exécuter un binaire à l'intérieur. C'est ce que dit précisément le titre de cette partie.
– essayez d'exploiter la vulnérabilité à nouveau. que remarquez-vous ?

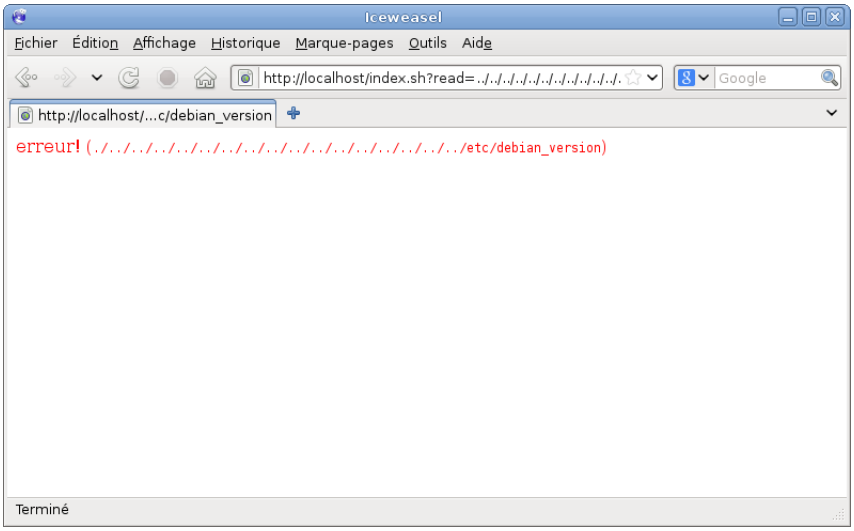

FIGURE 5 – Chroot jail manuelle - ../(../)*../etc/debian_version

Ca ne fonctionne plus, même en remontant très loin dans l'arborescence.

– le resultat attendu est-il celui escompté ? était-ce prévisible ? expliquez.
Oui, c'était prévisible. La racine fixée par la *chroot jail* empêche la remontée dans le système de fichier. Du coup, le fichier recherché n'existe tout simplement plus dans cette arborescence. On peut le constater aisément avec un appel à `strace` :

```
strace -f -e stat chroot . bin/thttpd -D -u root -c '**.sh' -nor -d var/www
stat("./(../)*../etc/debian_version", 0x7fffaeaf11a0) = -1 ENOENT
(No such file or directory)
```

– quels sont les avantages et inconvénients de cette méthode ? argumentez.
Avantages : le système de confinement est indépendant du binaire à confiner
· même si le binaire est corrompu ou vulnérable, le système reste fiable
· le mécanisme est réutilisable quelque soit le binaire à protéger
· le binaire n'en a pas conscience, et ne change pas son comportement
Inconvénients : il faut reconstruire un environnement complet pour le binaire
· cela peut être particulièrement long et fastidieux, il faut débugger
· des fichiers sont dédoublés sur le système (difficultés de maintenance)

– quelles seraient les conséquences potentielles d'une faille dans `rsyslogd` ?
Si `rsyslogd` est vulnérable, on pourrait imaginer que `thttpd` forge un message de log spécifique pour « sortir » de la *chroot jail* ; car en effet, le *daemon* de log n'est pas confiné. On appelle cela une « attaque par évasion ». Il faudrait donc, au choix, soit confiner `rsyslogd` aussi, soit ne pas l'utiliser du tout à l'intérieur, et faire logguer `thttpd` dans un fichier à la place (avec l'option `-l` prévue à cet effet).

4 La chroot jail applicative

– relancez le *daemon* avec ces nouveaux paramètres (hors *chroot jail*) :

```
thttpd -D -u root -c '**.sh' -r -dd var/www
```

– expliquez les nouveaux paramètres passés à **thttpd** en ligne de commande.
L'option **-r** utilise le mécanisme de *chroot jail* applicatif.
– en quoi reflètent-ils le titre de cette dernière partie du TP ? expliquez.
Réponse identique à la question précédente.
– vérifiez que l'option **-r** fait effectivement ce qu'elle prétend faire. (indice : **chroot** n'est pas seulement une commande, mais aussi le nom du *syscall*)

```
strace -e chroot thttpd -D -u root -c '**.sh' -r -dd var/www
chroot("$wwwroot/") = 0
```

– essayez d'exploiter la vulnérabilité à nouveau. que remarquez-vous alors ?

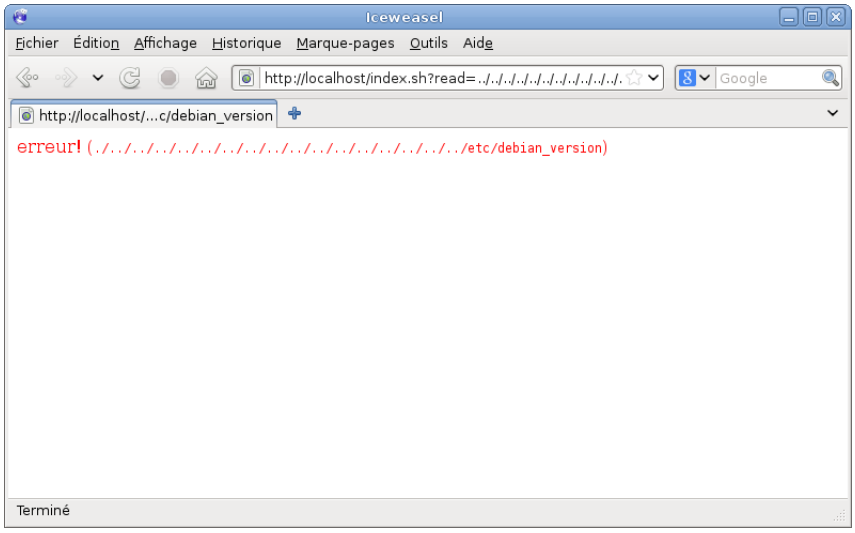

FIGURE 6 – Chroot jail applicative - ../(../)*../etc/debian_version

– le resultat attendu est-il celui escompté ? était-ce prévisible ? expliquez.
En pratique, l'effet est identique à une *chroot jail* manuelle.
– quels sont les avantages et inconvénients de cette méthode ? argumentez.
Ce mécanisme dispose des mêmes inconvénients que dans méthode manuelle, et ne garde aucun de ses avantages. Le succès de la méthode dépend maintenant du fait que le binaire réalise bien les opérations attendues (s'il est vulnérable c'est difficile à garantir). De plus, c'est une chance que **thttpd** propose cette fonctionnalité, mais c'est relativement rare, et de nombreux binaires ne le proposent pas nativement.
– en quoi cette méthode a un côté paradoxal ? rappelez-vous le but initial.
Comme expliqué brièvement ci-dessus, il est paradoxal de faire confiance à un binaire pour se chrooter sur une racine différente, alors qu'on cherche justement à le confiner en cas d'exploitation d'une vulnérabilité. Par analogie, cela revient un peu à donner à un détenu les clefs de sa cellule ! Mais c'est un autre débat.

À quel(s) autre(s) *daemon*(s) appliqueriez-vous cette méthodologie ? pourquoi ?

On peut l'appliquer à tout *daemon* qui accède au système de fichier : **ssh**, **ftp**, etc.