TP 1 : Environnement cloisonné

Romain CARRÉ
romain.carre@cea.fr

L'objectif de ce TP est de démontrer l'intérêt de **confiner une application** non fiable dans un système de fichier restreint. Il se déroule en trois étapes principales : d'abord l'analyse globale du problème avec une exécution sans protection ; puis la mise en évidence du procédé de confinement avec une construction de *chroot jail* ; et enfin, la comparaison fonctionnelle de cette dernière avec l'utilisation d'une option d'auto-confinement implémentée par le binaire lui-même. Nous nous intéresserons ici en particulier à `thttpd`, un serveur web dit « léger » (`http://acme.com/software/thttpd/`).

Bien que les durées indiquées dans le titre de chacune des parties soient approximatives, elles peuvent vous donner une idée du temps attendu pour réaliser les différentes tâches mentionnées. Si vous êtes totalement coincés sur une question, ou une procédure particulière du TP, inutile d'attendre 1h avant de venir me le signaler : faites moi signe !

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

1 Environnement de travail (∼10min)

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

2 Utilisation sans protection (∼30min)

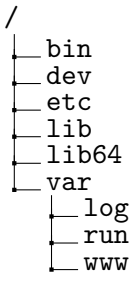
- lancez le *daemon* `thttpd` depuis le répertoire *wwwroot* avec la commande :

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

- à quoi servent les différentes options présentes sur la ligne de commande ?
- en quoi reflètent-elles correctement le titre de cette partie du TP ? expliquez.
- ouvrez un navigateur web, et chargez localement la page `index.sh` téléchargée.
- testez-la, par exemple en la faisant s'ouvrir elle-même. une idée de la faille ?
- parcourez son code source, mettez en évidence la vulnérabilité, exploitez-là.
- en tant qu'attaquant, quels fichiers intéressants consulteriez-vous grâce à elle ?
- en tant que professionnel de sécurité, que feriez-vous pour y remédier ?

3 Construction d'une cage (∼60min)

- construisez une *chroot jail* autour du *daemon* `thttpd`, avec :
  - les binaires `thttpd`, `sh`, `cut`, `grep`, et `cat` (à trouver)
  - toutes les bibliothèques qui leur sont dynamiques liées
  - deux fichiers de configuration qu'il faudra déterminer
  - un socket unix dans lequel `thttpd` peut envoyer ses logs
  - un fichier de log rempli progressivement par `rsyslogd`
  - le répertoire *wwwroot* qui contient les pages récupérées
  - un fichier *pidfile* qui contient le numéro de processus (utile quand on désire créer un script de démarrage)



Pour réaliser tout cela, vous aurez probablement besoin de lire les pages de manuel suivantes : `which`, `ldd`, `nsswitch.conf`, `passwd`, `rsyslog.conf` (notamment pour la directive `$AddUnixListenSocket` du module `imuxsock`, ou pour le filtre dynamique `:programname isequal`), `logger`, `thttpd`, et bien sûr `chroot`. Une partie de la documentation est parfois absente du système, il faut alors la chercher sur internet.

- relancez le *daemon* en vous plaçant à la racine de votre *chroot jail* :

```
chroot . bin/thttpd -D -u root -c '**.sh' -nor
```

- en quoi cette nouvelle commande reflète le titre de cette partie du TP ?
- essayez d'exploiter la vulnérabilité à nouveau. que remarquez-vous ?
- le resultat attendu est-il celui escompté ? était-ce prévisible ? expliquez.
- quels sont les avantages et inconvénients de cette méthode ? argumentez.
- quelles seraient les conséquences potentielles d'une faille dans `rsyslogd` ?

4 La chroot jail applicative (∼20min)

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

```
thttpd -D -u root -c '**.sh' -r
```

- expliquez les nouveaux paramètres passés à `thttpd` en ligne de commande.
- en quoi reflètent-ils le titre de cette dernière partie du TP ? expliquez.
- 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*)
- essayez d'exploiter la vulnérabilité à nouveau. que remarquez-vous alors ?
- le resultat attendu est-il celui escompté ? était-ce prévisible ? expliquez.
- quels sont les avantages et inconvénients de cette méthode ? argumentez.
- en quoi cette méthode a un côté paradoxal ? rappelez-vous le but initial.

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

5 Correction et questions (∼60min)

6 Annexes

```
#!/bin/sh

export PREFIX_DIR=./
export README_FILE=README.html

echo "Content-Type :text/html\n\n"

READ=$(echo $QUERY_STRING | grep -oE "(^|[?&])read=[^&]+" | cut -d= -f2)
if [ -z $READ ]; then READ=$README_FILE ; fi
FILEPATH=$PREFIX_DIR$READ

if [ ! -r $FILEPATH ]; then
    echo "<span style='color red;'>erreur ! (<tt>$FILEPATH</tt>)</span>" ;
else
if [ $FILEPATH = $PREFIX_DIR$README_FILE ]; then
    cat $FILEPATH ;
else
    echo "<textarea style='width :95%; height :95%;' readonly='readonly'>";
    cat $FILEPATH ;
    echo "</textarea>";
fi fi
```

Listing 1 – Page SH vulnérable