Pratique

Shells restreints – comment les détourner ?

Dawid Gołuński

Degré de difficulté

Les administrateurs, qui souhaitent restreindre la liberté des utilisateurs du système, décident de mettre en place un shell restreint, de sorte à rendre impossible l'utilisation de certaines commandes. Malheureusement, créer un environnement complètement restreint n'est pas facile et la moindre inattention risque de laisser à un utilisateur malin la possibilité de s'échapper.

Un shell restreint (*restricted shell*) est un shell système, modifiable de sorte à restreindre les possibilités des utilisateurs. Un tel shell est utilisé en général au moment où l'administrateur doit créer un compte pour un utilisateur à qui il ne fait pas complètement confiance. Il s'agit d'éviter des situations telles que collecter des informations sur le système et ses utilisateurs en chargeant les fichiers de configuration ou lancer les programmes système contenant des informations importantes.

Il est parfois aussi nécessaire de protéger l'utilisateur contre lui-même : un utilisateur débutant qui se sert d'un outil inconnu pourrait supprimer une partie de fichiers stockés sur son compte ou bien rendre disponible le fichier aux personnes non autorisées, en leur donnant des droits inadéquats.

Le serveur de messagerie, où le serveur Web ne fonctionne pas, peut constituer un exemple d'application d'un shell restreint. Le panneau de messagerie où les utilisateurs pourraient regarder leurs messages à distance (via un navigateur Web) n'existe pas alors. L'administrateur pourrait donc attribuer un shell restreint aux utilisateurs de sorte à lancer le client de messagerie et le programme *passwd* après une connexion distante pour modifier le mot de passe du compte.

Créer un environnement restreint

Afin de créer un environnement restreint, les administrateurs utilisent en général des shells

Cet article explique...

- Ce qu'est et comment fonctionne un shell restreint.
- Comment est structuré un environnement restreint typique.
- Vous connaîtrez les méthodes pour quitter un shell restreint.
- Ce qu'il faut utiliser pour créer un environnement restreint plus fiable.

Ce qu'il faut savoir...

- Connaître le travail avec le shell du système.
- Connaître au moins les bases du fonctionnement des systèmes du groupe linux/unix.
- Connaître les bases du langage C.

Erreur dans le programme bash

Créer un lien *rbash* et le paramétrer comme shell ne donne pas le résultat attendu sur certains systèmes : le shell ne commence pas à travailler en mode restreint. C'est une erreur qui a été corrigée dans la version 3.0 du programme *bash*. Il est également possible d'éviter ce problème en créant un script direct :

```
#!/bin/bash
/bin/rbash -l
```

Il faut enregistrer ce script par exemple en tant que */bin/rshell*, lui donner le droit d'exécuter et ensuite, le paramétrer comme un shell. Une fois connecté, vous devriez obtenir *rbash* qui fonctionne avec les restrictions. Il est toutefois conseillé aussi d'actualiser la version 3.0.

standard disponibles dans le système, tels que *bash*, *sh* ou *ksh*. Ces shells, appelés d'une manière adéquate, peuvent devenir des shells restreints. Il suffit en général d'appeler le fichier du shell sous le nom qui commence par la lettre r. Pour le *shell bash*, il s'agit de *rbash*, pour *sh – rsh* etc. Il est recommandé d'utiliser ici une liaison symbolique :

```
ln -s /bin/bash /bin/rbash
```

Une autre manière consiste à appeler le shell avec le paramètre `-r` ou `-restricted`. Le shell lancé sous ce mode appliquera des restrictions. Voici quelques restrictions les plus importantes :

- pas d'option de modifier le répertoire via la commande `cd`,
- l'utilisateur ne peut lancer que les commandes contenues dans la variable `PATH` ou intégrées dans le shell,
- pas d'option pour modifier les variables d'environnement `SHELL`, `PATH`, `ENV`, `BASH _ ENV`,
- il est interdit de lancer les commandes contenant le chemin du fichier (par exemple */bin/ls*),
- il est interdit de rediriger la sortie de l'application à l'aide des opérateurs `>` ou `>>`,
- la commande `exec` désactivée.

Une fois le shell restreint créé sous forme d'un lien symbolique – *rbash*, vous êtes prêts à l'attribuer à un utilisateur donné. Il est possible de le faire en éditant directement le fichier *passwd* ou via la commande :

```
usermod -s /bin/rbash john
```

Il est parfois nécessaire d'ajouter une ligne */bin/rbash* au fichier */etc/shells*, sinon une partie de services (par exemple, le serveur FTP) peut refuser de connecter l'utilisateur. Paramétrer le shell n'est pas tout ; l'étape suivante consiste à préparer le répertoire de l'utilisateur.

Ce répertoire devrait disposer des droits 511 (il est recommandé que son propriétaire soit root et non l'utilisateur). Créez le fichier *.bash_profile* dans le répertoire et paramétrez le droit d'accès 444 et le propriétaire *root*.

Mis à part le répertoire personnel de l'utilisateur, il faut ouvrir le répertoire dans lequel vous placerez les programmes sélectionnés. Il peut s'agir par exemple de */usr/local/rbin*. Vous placerez dans ce répertoire les liens symboliques à l'application que vous voulez rendre disponible, par exemple :

```
ln -s /usr/bin/passwd \
      /usr/local/rbin/passwd
```

La dernière étape consiste à éditer les fichiers de configuration du shell. Le fichier */etc/profile* contient les paramètres globaux.

Pour que les entrées de ce fichier n'influencent pas les paramètres du shell *rbash*, il faut placer son contenu dans l'instruction conditionnelle :

```
if [ "$0" = "-bash" ]; then
#saisissez ici le contenu actuel#
#du fichier /etc/profile #
fi
```

Le fichier local de paramètres *.bash_profile* doit en revanche contenir une seule ligne qui définit le chemin au répertoire avec les programmes préparés :

```
export PATH=/usr/local/rbin
```

Si vous voulez refuser à l'utilisateur la possibilité d'exécuter certaines commandes internes, il faut ajouter à la fin du fichier une ligne contenant la commande `enable -n`, après laquelle vous citez les commandes à désactiver. À titre d'exemple :

```
enable -n pwd unset
```

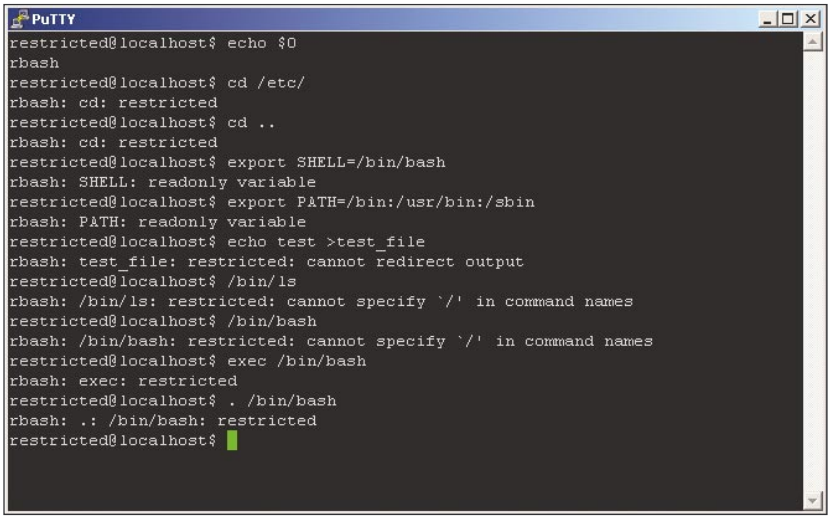

Figure 1. *Restrictions mises en place par le shell rbash*

Comment cela marche-t-il ?

Lorsqu'un utilisateur se connecte au système (*ssh*), il sera transféré au répertoire personnel ; ensuite, le shell */bin/rbash* se lancera. Le shell lit les fichiers de configuration en commençant par */etc/profile*. L'instruction conditionnelle placée au début vérifie si l'argument zéro (`argv[0]`) est égal à la suite de caractères `-bash`. En ce qui concerne le shell rbash, l'argument `0` est égal à `-rbash` et donc les commandes placées à l'intérieur de la condition if ne seront pas exécutées.

Le shell exécutera le fichier local du fichier de configuration – *.bash_profile*, qui paramètre la variable d'environnement `PATH`. Le fait de modifier cette variable peut surprendre car conformément à ce que nous avions écrit auparavant, l'une des restrictions du shell rbash est le paramétrage de la variable `PATH` en lecture. Ceci est dû au fait que toutes les restrictions citées ne sont mises en place qu'après le chargement des fichiers de démarrage (c'est pour cette raison qu'il est très important de paramétrer des droits adéquats).

Échapper au piège

L'environnement restreint créé d'après le schéma présenté semble être sûr. Malheureusement, la sécurité est étroitement liée aux programmes disponibles et aux services fonctionnant sur le serveur, qui à priori ne sont pas liés à l'environnement. Nous essaierons de vous présenter quelques exemples de situations où il est possible d'éviter les protections du shell.

Opérations sur les fichiers

Certains administrateurs oublient le fait que le shell restreint *rbash* n'enferme pas l'utilisateur dans un arbre séparé de répertoires (en utilisant la fonction `chroot()`) – il bloque seulement la modification du répertoire à l'aide de la commande `cd`. Ils oublient aussi que les appels aux fichiers contenant le chemin d'accès sont bloqués seulement quand l'utilisateur essaie d'appeler directement l'application, par exemple */bin/ls* ou d'exécuter l'instruction `exec /etc/profile` ou bien `source /etc/profile`. Cela signifie que si l'utilisateur dispose de l'accès au programme *pico*, rien ne l'empêche de voir le fichier *passwd* après avoir fait la commande : `pico /etc/passwd`.

Shell escape

La plupart des programmes n'ont pas hélas été écrits pour travailler avec les shells restreints. Certains d'entre eux contiennent une fonction intégrée, permettant d'appeler le shell (*shell escape*). Cette fonction est en général appelée à l'aide du caractère *!*. C'est utile lorsque vous travaillez sous contrôle du client FTP et vous souhaitez consulter la liste de fichiers dans le répertoire courant. Vous n'êtes pas alors obligé d'ouvrir une nouvelle session SSH, il suffit de faire la commande : `! ls -l` dans la ligne de commandes. Quelques exemples de programmes standard disponibles dans le système et proposant le *shell escape* : *telnet*, *mail*, *ftp*, *less*, *more*, *vi*, *gdb*, *lynx*. Nous pouvions nous attendre à ce que les commandes transférées soient exécutées en se basant sur le shell défini dans la variable `SHELL` – dans notre cas */bin/rbash*. Hélas, ce n'est pas toujours le cas. Une partie de programmes ne tient pas compte des variables d'environnement et ils exécutent les commandes transférées à l'aide du shell */bin/bash* ou bien */bin/sh* défini en permanence (dans le code du programme). Le programme

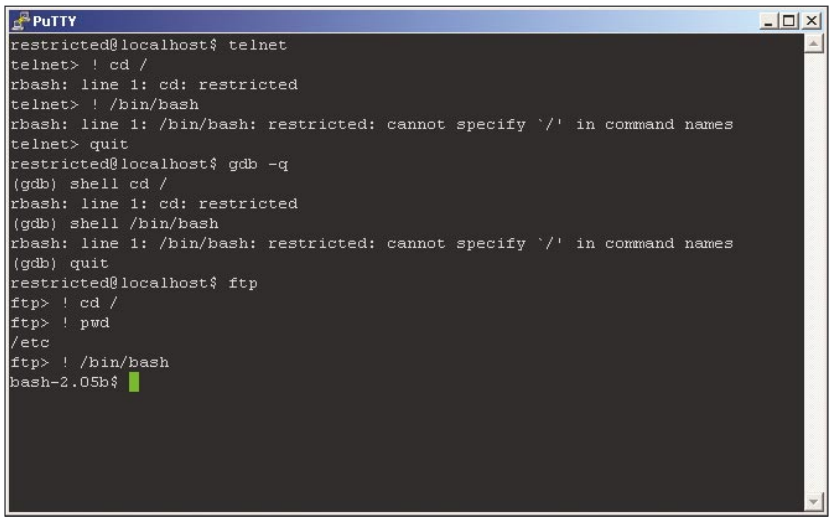

Figure 2. *Utilisation d'une fonction shell escape dans les programmes telnet, gdb et ftp*

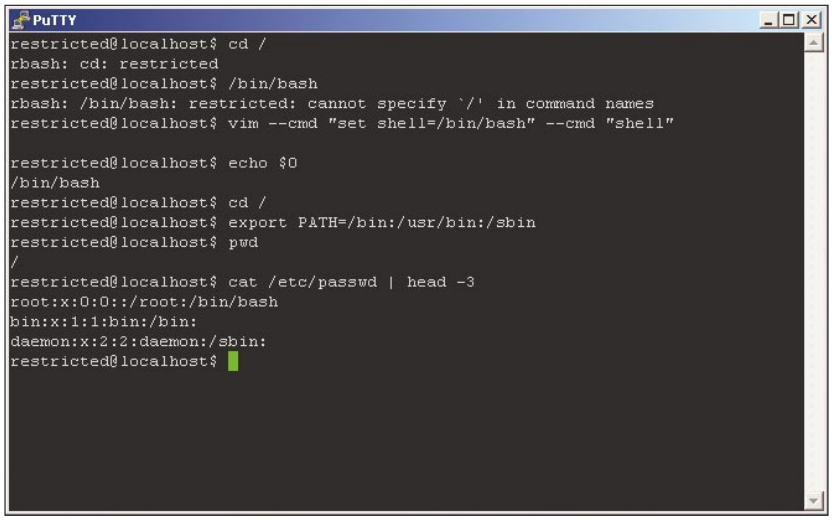

Figure 3. *Lancement du shell /bin/bash à l'aide de la fonction shell de l'éditeur vim*

vim peut constituer ici un exemple intéressant. Il est vrai qu'il tient compte de la variable d'environnement `SHELL` ; après avoir commencé à fonctionner, il copie son contenu dans la variable interne shell. Il existe toutefois un problème : il est possible d'éditer à votre gré les paramètres du programme *vim* lors de son fonctionnement. Il suffit pour ce faire d'effectuer deux commandes dans la ligne de commandes (pour y accéder, vous disposez de la séquence de touches [*ESC*]+[:]) :

```
set shell=/bin/bash
shell
```

Il est également possible de faire ces deux commandes au moment d'appeler l'éditeur *vim*. Vous disposez du paramètre `--cmd`, ce qu'illustre la Figure 3. Vous obtiendrez en résultat le shell *bash*, sans restrictions. Soulignons ici que *vim* permet de bloquer les possibilités d'utiliser le shell. De même que dans le cas de *bash*, il suffit de créer un lien symbolique appelé *rvim*. Désormais, chaque tentative d'exécuter la commande shell se termine par le message suivant :

```
Shell commands not allowed in rvim
```

Liens aux programmes d'aide

Certains programmes utilitaires, dont l'objectif consiste à exécuter certaines opérations, se réfèrent aux applications externes, par exemple, l'éditeur populaire *pico* se réfère au programme *ispell*, afin de vérifier si le texte édité est correct, les navigateurs *links* et *lynx* appellent le programme de messagerie à chaque fois que vous ouvrez le lien commençant par `mailto:` etc.

Le problème apparaît au moment où l'utilisateur peut définir le chemin du programme. Rien ne l'empêche alors de donner le chemin à une autre application au lieu du programme ispell. En fonction de la manière utilisée pour lancer le programme externe (y compris, les liens utilisés), il est possible de lancer correctement l'application choisie, par exemple, le programme bash. En ce qui concerne le programme pico, c'est simple. Vous pouvez modifier le chemin du programme ispell à l'aide de `-s` :

```
pico -s /bin/bash
```

Une fois l'éditeur lancé, saisissez les commandes que vous souhaitez transférer au shell *bash*, en les séparant avec les caractères d'une nouvelle ligne. Terminez l'édition à l'aide des touches [*CTRL*]+[*T*], correspondantes au transfert de la mémoire tampon de l'édition au programme défini. Ce transfert se déroule de manière suivante : dans un premier temps, la mémoire tampon est enregistrée dans un fichier temporaire et ensuite, le chemin de ce fichier est transféré en argument :

```
/bin/bash /tmp/pico.xxxx
```

Le résultat d'une commande ainsi créée consiste à interpréter le fichier temporaire *pico* comme s'il était un simple script *bash*. De nombreux programmes permettent de définir un éditeur texte favori (par exemple *pine*). Même si vous ne voyez pas les possibilités d'une telle modification dans la configuration du programme ni dans les commutateurs disponibles, il faut vérifier si le programme n'utilise pas la variable d'environnement appelée `EDITOR` ou `VISUAL`. Ces variables ne sont pas

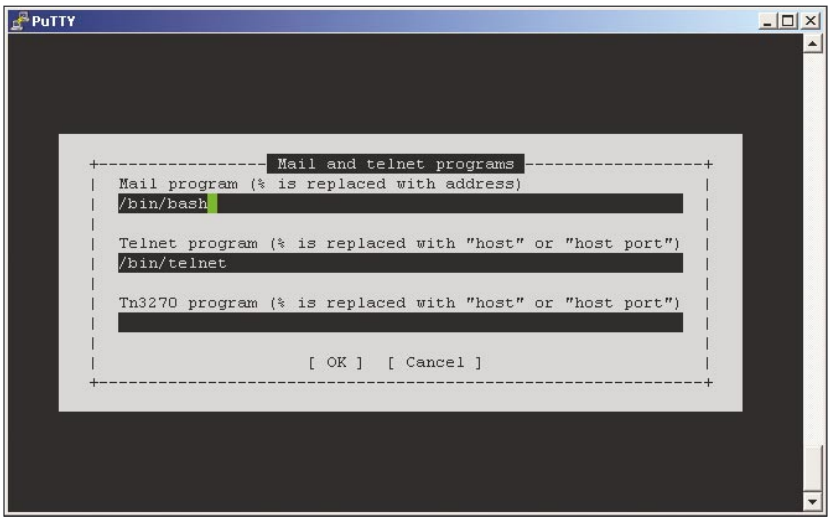

Figure 4. *Modification du chemin du programme qui supporte le protocole mail dans le programme links*

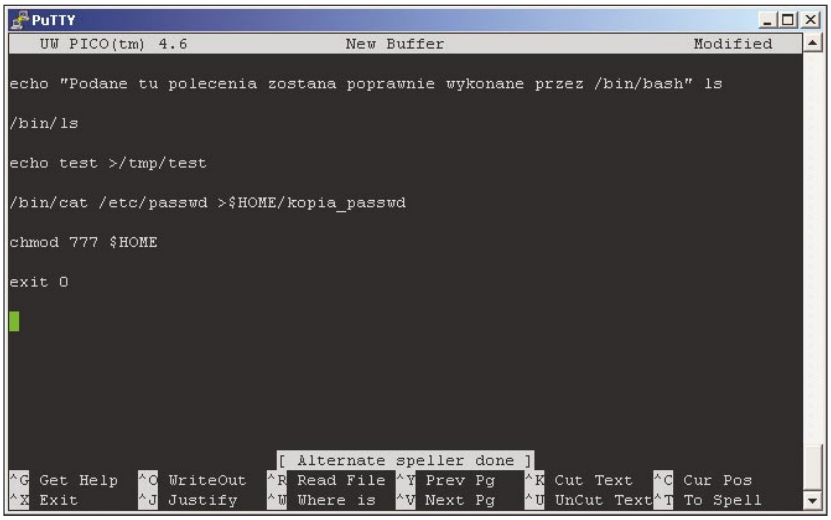

Figure 5. *Utilisation de la fonction qui vérifie l'orthographe en pico pour faire les commandes du shell bash*

protégées par les restrictions de rbash et il est donc possible d'écrire n'importe quel chemin.

Remplacement de la variable $SHELL

Si le serveur SSH fonctionne sur le serveur et si l'option `Permit UserEnvironment` du fichier de configuration des services *sshd_config* est paramétrée à `ON` (cette option est activée par défaut), certaines variables d'environnement peuvent être modifiées, ce qui peut influencer le fonctionnement de *rbash*. C'est possible parce que le *deamon sshd* paramètre les variables d'environnement élémentaires après avoir authentifié l'utilisateur et ensuite, il charge le fichier *.ssh/environment*, si ce dernier existe. Supposons que ce fichier contient la ligne :

```
SHELL=/bin/bash
```

Que se passe-t-il alors ? Une fois l'utilisateur connecté, le *deamon SSH* paramètre la variable `SHELL` à la valeur */bin/rbash* en se basant sur les informations du fichier *passwd* ; après la lecture du fichier environment, la variable sera toutefois écrasée.

Afin d'effectuer une telle attaque, il est nécessaire (mis à part la configuration susmentionnée de *sshd*) que le répertoire *.ssh* (éventuellement, l'accès à la com-mande `mkdir`) existe avec l'option d'enregistrement.

Lorsque le répetoire personnel dispose des droits `+rx`, vous pouvez d'une manière simple lister son contenu via la commande (intégrée) `echo *`.

L'accès à l'application, permettant de créer le fichier environment, sera également nécessaire. Vous n'êtes pas obligés d'opter pour un éditeur texte, un client populaire pine s'y prêterait parfaitement.

L'opération consiste à envoyer le fichier *environment* par mail d'un compte différent, à recevoir le courriel et à utiliser la fonction du programme *pine*, qui permet d'enregistrer le contenu du courriel dans l'emplacement précisé. Cette fonction est disponible après que vous ayez choisi l'option *ViewAttach*. Si l'enregistrement réussit, la variable `SHELL` devrait être modifiée après la reconnexion au compte. Grâce à cette démarche, il sera possible d'utiliser la fonction susmentionnée *shell escape* d'un des programmes disponibles, par exemple *telnet*, afin d'exécuter les commandes en vous basant sur le shell défini.

.forward

Le fichier *.forward* placé dans le répertoire personnel de l'utilisateur est utilisé par le programme *sendmail* pour rediriger les courriels (destinés à l'utilisateur) à l'adresse indiquée. Mis à part les adresses de messagerie électronique, il est autorisé de préciser le chemin du programme auquel vous voulez envoyer le contenu du courriel. Puisque vous disposez de l'option d'éditer ce fichier, paramétrez-le à :

```
john@mailserver.com, "| /bin/bash -c
      notre commande"
```

Le courriel arrivant sera envoyé à l'adresse de John et aussi à l'entrée du programme *bash*, ce qui ferra exécuter votre commande. Heureusement, l'attaque de ce type échouera dans la plupart de cas parce que la commande donnée est le plus souvent exécutée en se basant sur le shell restreint spécial de *sendmail* appelé *smrsh*. Ce shell fait

Listing 1. *Script .bash_profile développé*

```
echo -n "Bienvenue sur le serveur"
echo '/bin/hostname'
echo
echo "Veuillez prendre connaissance des règles : "
cat /etc/motd
sleep 5
clear
export PATH=/usr/local/rbin
enable -n pwd
```

Listing 2. *Bibliothèque partagée qui lance le shell bash*

```
#include <stdio.h>
void localtime() {
   unsetenv("LD_PRELOAD");
   system("echo ok! && /bin/bash");
   exit(0);
}
```

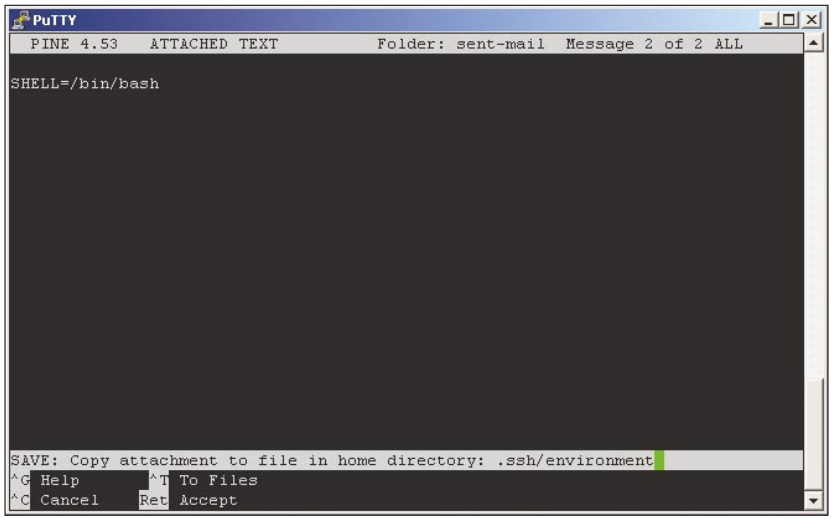

Figure 6. *Utilisation du programme pine pour enregistrer le fichier environment*

en sorte que seuls les programmes contenus dans le répertoire */etc/smrsh* puissent être exécutés.

Modifier les droits des fichiers et des répertoires

Comme vous l'avez déjà constaté, les droits attribués aux fichiers et aux répertoires contenus dans le répertoire personnel sont extrêmement importants.

Si l'administrateur se trompe et rend disponible la commande `chmod`, il se peut que l'utilisateur attribue le droit `+w` au répertoire personnel, ce qui l'autorisera à supprimer le fichier *.bash_profile*. À la prochaine connexion, la variable `PATH` ne sera pas restreinte à */usr/local/rbin* et donc il est fort probable que le chemin contiendra le répertoire */bin* (ce qui permettra de lancer le *bash* normal). Il est parfois possible de modifier les droits sans utiliser le programme */bin/chmod*. Imaginez que le service FTP fonctionne sur ce serveur.

Si l'administrateur n'a pas restreint l'accès à ce service et n'a pas fait de restriction sur l'instruction `FTP SITE CHMOD`, l'utilisateur peut faire la commande `chmod 777` après s'être connecté au port *21* pour supprimer facilement le fichier *.bash_profile* ou l'écraser avec un autre.

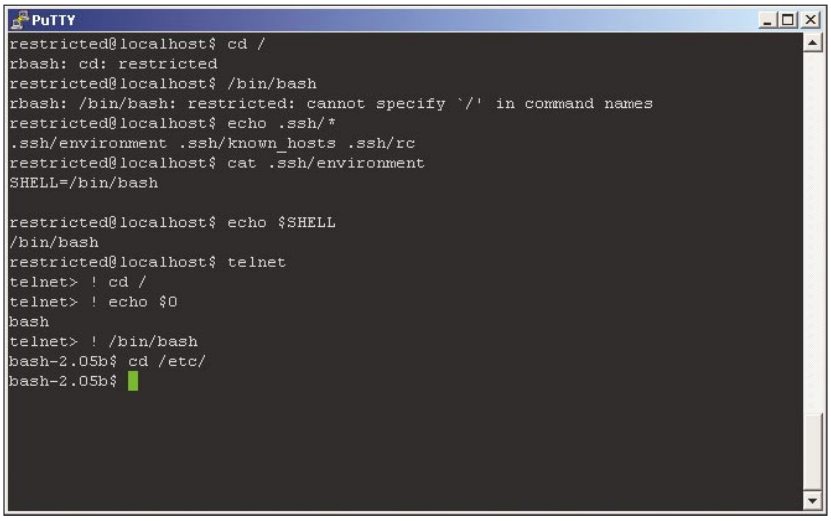

Figure 7. *Détourner les restrictions en écrasant $SHELL avec le fichier .ssh/environment*

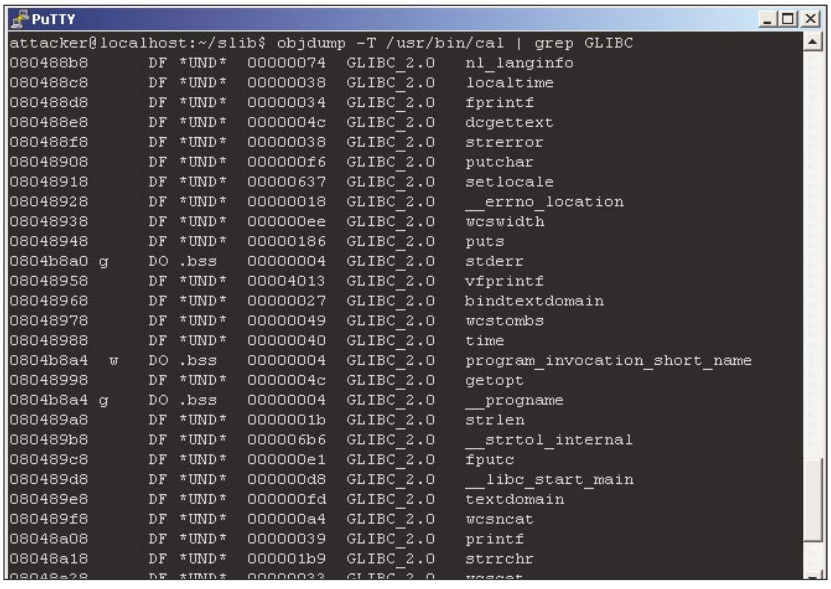

Figure 8. *Tableau de symboles du programme cal*

Interrompre le script de démarrage

Les administrateurs créent souvent des scripts de démarrage développés et oublient la gestion de signaux. Regardons le script du Listing 1. Il suffit que l'utilisateur appuie sur les touches [*CTRL*]+[*C*] lors de la commande `sleep 5` et le signal `SIGINT` sera envoyé. Le script s'arrêtera alors immédiatement. Si l'exécution de la commande export échoue, la variable `PATH` ne sera pas paramétrée et l'utilisateur ne sera pas limité aux commandes du répertoire *rbin*. Les signaux peuvent être contrôlés à l'aide de la fonction `trap`. Pour ignorer le signal `SIGINT`, faites la commande :

```
trap "" SIGINT
```

Téléchargement de la bibliothèque

Une méthode intéressante pour quitter le shell restreint, capable de réussir, consiste à charger la bibliothèque spécialement préparée. C'est possible parce que le *linker/loader* (*ld.so*) du système permet de définir la bibliothèque partagée qui sera chargée avant l'exécution du programme. Il existe alors plus de chances pour écraser la fonction choisie, utilisée par le programme lancé (par *loader*).

Il existe deux manières pour indiquer le chemin de la bibliothèque que doit charger *linker* : l'enregistrer dans le fichier */etc/ld.so.preload* ou dans la variable d'environnement `LD_PRELOAD`. Seul le *root* a l'accès au fichier *ld.so.preload* ; il faut se servir de la variable. Le shell *rbash* ne met pas cette variable en mode *readonly*. Si l'administrateur n'a pas bloqué toutes les commandes qui servent à modifier l'environnement (ce qui arrive rarement), nous serons capables d'y attribuer n'importe quel chemin.

Mis à part l'accès à la commande qui modifie l'environnement, il faut disposer de l'endroit et de la manière à enregistrer le fichier binaire de la bibliothèque. Si vous disposez du droit d'enregistrer dans le répertoire personnel (ou l'un de sous-répertoires) et de l'accès FTP, il n'y a plus de problèmes. Si vous disposez de l'accès au

programme *pine*, vous pouvez envoyer la bibliothèque compilée dans la pièce jointe du message électronique. Le programme *pine* ne donne pas le droit d'exécuter les pièces jointes sauvegardées, ce qui n'empêche pas de charger la bibliothèque dynamique via *ld.so* (il suffit que le fichier ait le droit `+r`). Il reste encore le problème de l'emplacement pour enregistrer le fichier.

Même si vous ne disposez pas de droit `+w` dans le répertoire personnel, vous avez toujours la possibilité de réussir vos opérations. Vous pouvez utiliser l'un des répertoires destinés aux fichiers temporaires, */tmp*, ou */var/tmp* (même s'il est monté en tant que *noexec*). Imaginons toutefois un scénario restreint : vous n'avez pas d'accès FTP, de droit d'enregistrement dans le répertoire personnel et le seul programme attribué est un simple calendrier : *cal*. Il est toujours possible de vous en sortir. Puisque vous vous êtes connecté au système via SSH, la manière la plus simple consiste à copier votre fichier binaire dans le répertoire */tmp* à l'aide de *scp*.

Il reste à écrire la bibliothèque partagée. Puisque l'attaque consistera à écraser une des fonctions du programme attaqué par votre code, il faut déterminer quelles fonctions il utilise avant de créer la bibliothèque. Il faut analyser le seul programme auquel vous pouvez accéder, ici : *cal*. Afin de connaître les fonctions utilisées par ce programme, vous pouvez afficher le tableau de symboles à l'aide du programme *objdump* de manière suivante :

```
objdump -T /usr/bin/cal | grep GLIBC
```

Vous remarquerez que l'une des fonctions utilisée est `localtime()`. Avec cette information, vous pouvez commencer à écrire le programme de la bibliothèque. L'exemple de code du langage C présenté sur le Listing 2 effectue quatre opérations : supprimer la variable `LD_PRELOAD` de l'environnement (afin d'éviter d'entrer dans les programmes exécutés), afficher le texte *ok!*, lancer le programme du shell */bin/bash* et interrompre le fonctionnement du programme (*cal*) à la fin du travail avec le shell. La compilation doit être effectuée de manière suivante :

```
gcc -shared slibrary.c -o slibrary
```

Après avoir envoyé la bibliothèque sur le serveur, vous êtes prêt à passer à la partie essentielle de l'attaque. Une fois connecté, paramétrez la variable d'environnement `LD_PRELOAD` via la commande declare ou *export*. Nous supposons ici que la bibliothèque se trouve dans le répertoire *tmp* :

```
declare -x LD_PRELOAD=/tmp/slibrary
```

Ensuite, lancez le calendrier en saisissant *cal*. Si tout va bien, le programmer appellera lors de son fonctionnement la fonction `localtime()` où se trouve votre code. Vous devriez voir s'afficher le texte *ok!* et l'invite de commande du nouveau shell.

Comment se protéger ?

Les exemples présentés des attaques démontrent qu'un seul programme mal choisi (proposant une fonction en apparence ordinaire en tant que paramètre de l'éditeur favori) peut permettre de quitter le shell restreint *rbash*. Est-il possible de faire en sorte que l'environnement restreint soit plus fiable ? Deux projets *open source* nous viennent en aide.

Le premier d'entre eux jail sert à créer un environnement séparé où sera enfermé l'utilisateur après sa connexion. Cet environnement est créé à l'aide de la fonction `chroot()`. Le processus fonctionnant dans l'environnement `chroot` est limité au répertoire défini et n'a pas d'accès à l'arbre principal. Ce processus perçoit le répertoire où il est enfermé comme un répertoire principal : *root* (*/*).

Vous obtenez ainsi plus de sécurité car l'utilisateur est incapable de sortir en dehors de l'arbre attribué au processus et n'a l'accès qu'aux fichiers créés dans son cadre. Ce point concerne tous les fichiers, y compris les bibliothèques, les programmes et les fichiers de périphériques. Le projet *jail* facilite considérablement la collection de tous les fichiers, nécessaires pour que l'environnement puisse fonctionner.

Le second projet s'appelle *Iron Bars Shell – restricted system shell for Linux/Unix*. C'est un projet du shell restreint, appelé *ibsh*. Il est créé en vue de renforcer la sécurité. L'une de principales idées du projet consiste à interdire toutes les opérations qui n'ont pas été définies comme autorisées. Le shell propose aussi plusieurs

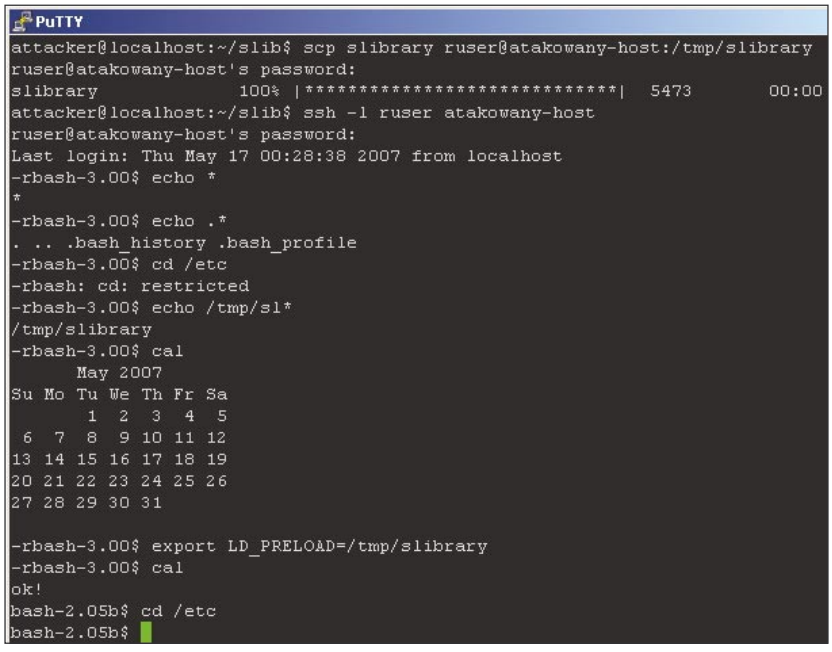

Figure 9. *Quitter l'environnement restreint grâce au chargement de la bibliothèque*

fonctions intéressantes, comme le contrôle des paramètres des commandes exécutées, la connexion des activités de l'utilisateur au *syslog* ou bien le blocage de l'accès aux fichiers dont l'extension ne se trouve pas sur la liste d'extensions autorisées, créée par l'administrateur (paramétrer le blocage aux fichiers **.c* empêche la compilation des sources potentiellement dangereuses C, etc.). Vous obtiendrez ainsi un contrôle plus large sur les opérations de l'utilisateur.

Conclusion

La pratique démontre qu'il est difficile de créer un environnement restreint, ne se basant que sur le shell restreint. Il faut prévoir toute opération que pourrait effectuer l'utilisateur et approfondir les principes du fonctionnement de chaque programme disponible en fonction des options proposées.

Sélectionner les programmes peut poser des problèmes parce que, comme vous l'avez constaté, même un simple éditeur de texte peut contenir des fonctions permettant d'accéder au shell. Nous pouvons conclure que le shell restreint ne peut être utilisé qu'en complément des techniques plus efficaces de restrictions des utilisateurs (comme chroot) ou éventuellement, en tant que shell pour les utilisateurs débutants du système. ●

À propos de l'auteur

L'auteur est autodidacte, passionné d'informatique et notamment de sécurité depuis de nombreuses années. Il étudie les réseaux informatiques dans le cadre du programme *Cisco Network Academy* de l'école polytechnique de Poznań. Contact avec l'auteur : *golunski@crackpl.com*

Sur Internet

- *http://www.jmcresearch.com/projects/jail/* – Site officiel du projet Jail,
- *http://ibsh.sourceforge.net* – Site officiel du projet Iron Bars Shell.