par Mulyadi Santosa
<a_mulyadi/at/softhome.net>

L´auteur:

Bonjour, mon nom est Mulyadi Santosa. Je vis en Indonésie et travaille en freelance comme auteur et consultant. Mes centres d'interêt en informatique sont les clusters, la sécurité système et les réseaux. J'ai aussi monté une petite société spécialisée dans les solutions de clustering "clés en main" : OpenMosix et OpenSSI ont énormément captivé mon attention. Côté loisirs, je suis un grand fan de lecture et de sport. Si vous voulez discuter de tout cela plus en détail, envoyez moi un mail à a_mulyadi@softhome.

Traduit en Français par:
Guillaume Baudot
<guillaume.baudot(at)caramail.com>

Suivre les rouages internes de Linux avec Syscalltracker

Résumé:

Il arrive parfois que l'on veuille voir précisément ce qui se passe dans notre sytème Linux, et nombre de programmes sont à notre disposition pour enregistrer les activités (logs), détecter des intrusions, vérifier l'intégrité du système, etc. Aujourd'hui, je voudrais vous présenter un programme qui permet d'observer l'activité au niveau du noyau, technique qui a pour mérite d'être à la fois fiable et généraliste.

Introduction

Captivé un jour par une discussion dans une liste de diffusion traitant de clusters, je suis accidentellement tombé sur le sujet suivant : une personne rapportait avoir rencontré une anomalie système suite à l'application d'un patch au noyau. Quelqu'un est alors intervenu pour lui proposer de reconstituer les étapes du problème. Et pour isoler la défaillance, il a fait appel à Sycalltracker. Après une telle performance, je me demandai aussitôt

: "Mais quel genre d'outil peut−ce bien être ? À quoi sert il ?". Pour le novice que j'etais alors, le nom "Syscalltracker" à lui seul me semblait empreint du plus sombre mystère :−)

Syscalltracker

Le projet http://syscalltrack.sourceforge.net consiste en une suite de modules pour le noyau Linux, destinés à intercepter les appels système en provenance de ce dernier. Mais encore ?! Et bien, on l'utilise généralement afin de résoudre des problèmes de dysfonctionnement pour lesquels les outils plus classiques de traçage ou de déboguage sont inefficaces. Pour une meilleure compréhension, laissez−moi vous donner un exemple : supposons que vous ayez un fichier /etc/inetd.conf (fichier de configuration du démon inetd), paramètré de façon à lancer plusieurs démons, en désactiver d'autres... Tout semble aller pour le mieux dans le meilleur des mondes quand soudain, le fichier de configuration disparaît sans raison apparente. Fort heureusement, vous ètes quelqu'un de prévoyant et avez effectué une sauvegarde : vous vous empressez donc de rétablir le fichier, puis de relancer inetd. Et cette fois, ce sont de nouvelles lignes qui s'insèrent mystèrieusement dans /etc/inetd.conf... Diantre, il y a de quoi être déstabilisé : "Est−ce dû à une intention maligne ou un effet de bord du démon inetd ? Suis−je victime d'une attaque susceptible de compromettre mon système ?", autant de question qu'il est déplaisant de laisser en suspens... Et vous avez beau passer en revue les fichiers log du système, analyser méticuleusement les résultats des commandes "top" ou "ps", rien n'y fait, vous n'avez toujours pas la moindre piste !

Syscalltracker est la solution à ce genre de problème. Je n'irai pas jusqu'à dire que c'est la panacée, mais dans la majorité des cas, il devrait vous aider à vous tirer d'affaire. Le principe sous−jacent est que toute opération réalisée par quelque processsus que ce soit, le sera par le biais d'une procédure interne (voir plusieurs) propre au système : on parle communément d'*appel système*. Concrètement, si vous supprimez un fichier, il est fait appel à *unlink()*, si vous lancez un script shell, il s'agit de *exec()* ou *execv()*. Donc en pratique, toute action liée au système se concrétise par un *appel système*. Et comme la finalité de Syscalltracker est justement d'intercepter les appels à ce type de procédure, cela en fait un outil d'analyse redoutablement efficace.

Motivé(e) ?

Alors, c'est le moment d'essayer : commencez par télécharger l'archive sur le site http://syscalltrack.sourceforge.net. Pour cet article, j'ai utilisé syscalltrack−0.82.tar.gz (500 Ko env.) sur un système Redhat Linux 7.3 : n'oubliez pas d'adapter l'exemple à votre cas ! Ceci dit (et cela fait), décompressez le fichier, disons dans /usr/src.

#cd /usr/src && tar xzvf syscalltrack−0.82.tar.gz

Assurez vous ensuite que vous disposez du code source du noyau dans /usr/src, c'est indispensable !

rpm −qa | grep −i kernel−source

ou

ls −al /usr/src/linux−2.4

Donc, si ce n'est pas le cas, il va falloir corriger le tir. Vous pourrez trouver les sources du noyau Redhat sur le CD2 :

rpm −replacepkgs −Uvh /path/to/your/RPM/kernel−source−2.4.18−3.i386.rpm

Attention : Syscalltracker doît être compilé spécifiquement pour la version du noyau (rustines comprises) que vous utilisez. Ainsi, pour le noyau standard de la Redhat 7.3, il vous faut en effet les sources présentes sur le CD2 (kernel−source−2.4.18−3.i386.rpm), mais si vous basculez vers un autre noyau, vous devrez alors recompiler Syscalltracker pour ce même noyau.

En plus du code source du noyau, vous avez encore besoin de son fichier de configuration. Voyez d'abord dans /boot s'il est présent (s'il s'agit d'une partition séparée, vérifiez qu'elle est montée).

ls −al /boot/config*

Si la sortie indique entre autres fichiers quelque chose comme 'config−2.4.18−3', c'est celui−ci qu'il faut copier.

cp /boot/config−2.4.18−3 /usr/src /linux−2.4/.config

Si le fichier vient à manquer, on peut encore le trouver parmi les sources du noyau dans le répertoire *configs*. Pour connaître avec certitude votre version du noyau (et ainsi ne pas vous tromper de fichier de configuration), examinez le résultat de la commande suivante :

uname −a

La version du noyau est affichée. Si vous lisez "kernel−2.4.18−3−i386", c'est sans aucun doute *kernel−2.4.18−3−i386.config* que vous copierez.

cd /usr/src/linux−2.4
cp configs/kernel−2.4.18−3−i386.config ./.config

N.d.T. : Pour d'autres versions de Linux, vous ferez plutôt :

```
# cd /usr/src/linux−<your_version>
# cp arch/<your_arch>/defconfig .config
```

La dernière étape :

```
#cd /usr/src/linux−2.4.18−3
# make mrproper
# make menuconfig
```

Modifiez les options suivant vos besoins, sauvegardez et quittez. Si vous avez compilé vous−même votre noyau, je vous souhaite de n'avoir pas égaré son fichier de configuration : sa reconstitution à l'identique serait un vrai casse−tête :−)
N.d.T. : il est précisé sur le site (et dans les commentaires sur l'article) que le noyau et Syscalltracker doivent être compilés avec la même version de gcc, du fait de la nature du programme (modules noyau). D'aucuns pourraient donc être tentés de commencer par compiler leur propre noyau : c'est d'autant moins difficile que l'essentiel de la procédure est décrit au dessus et que l'ordinateur se charge du reste en quelques commandes.

Compiler Syscalltracker

Les prérequis étant réglés, nous pouvons dorénavant nous consacrer à Syscalltracker.

```
# cd /usr/src/syscalltrack−0.82
# ./configure (ou encore ./configure −with−linux=/path/to/your/linux/kernel/source)
# make && make install
```

Après compilation, vous aurez à votre dispostion deux nouveaux modules :

1. /lib/modules/2.4.18−3/syscalltrack−0.82/sct_rules.o
2. /lib/modules/2.4.18−3/syscalltrack−0.82/sct_hijack.o

Ces modules vont nous permettre d'analyser notre système en utilisant une technique qualifiée de *détournement d'appel système* par l'auteur lui−même : tout appel système est intercepté, provoquant au besoin une action pré−définie (par nos soins) avant exécution de la procédure normale. Pour les charger dans le système, logez vous sous le compte "root" et utilisez ce script qui se charge des détails pour vous :

```
# sct_load
```

Pour vous assurer du chargement des modules, voyez avec *lsmod* :

```
# lsmod
Module     Size  Used by    Not tainted
[...]
sct_rules  257788   2
sct_hijack 110176   1      [sct_rules]
[...]
```

Les règles

Félicitations ! L'installation est terminée, les modules chargés, donc Syscalltracker est d'ores et déjà fonctionnel. Mais ce n'est pas tout, il nous reste à voir comment écrire les *règles* qui détermineront le comportement du programme. Pour se mettre en jambe, voici un premier exemple tout simple :

```
rule
{
   syscall_name=unlink
   rule_name=unlink_rule1
   action
   {
     type=LOG
     log_format {%comm : %params suppression par %euid --> %suid}
   }
   when=before
}
```

La syntaxe est plutôt limpide : chaque règle débute par le mot−clé "rule", suivi d'un accolade ouvrante ("{"). Il faut ensuite déclarer à quel appel système elle s'applique en renseignant le paramètre "**syscall_name**". Pour une liste complète des appels système, voyez le contenu du fichier '/usr/local/lib/syscalltrack−0.82/syscalls.dat−2.4.18−3' (ou ...syscalls.dat−<your_linux_version>). Pour notre exemple, j'ai choisi *unlink()* : la règle s'appliquera chaque fois qu'un utilisateur ou un programme tentera d'effacer un fichier.

Le paramètre "**rule_name**" vous permet d'identifier la règle de façon unique. Ce choix est entièrement libre, mais je vous recommande chaudement de faire simple et compréhensible pour votre propre confort ! J'ai personnellement choisi "**unlink_rule1**". La partie "**action**" (entre accolades) définit l'action à réaliser quand la règle est vérifiée. Syscalltracker connaît différentes actions, mais je m'en tiendrai pour l'instant au type **LOG**, qui transmet un compte−rendu d'activité vers **/dev/log**. À en croire le site WEB, il est prévu dans les améliorations que l'on puisse redéfinir les appels système, par exemple pour en modifier les paramètres :−)

Quand on sélectionne une action de type **LOG**, on précise ensuite le format d'enregistrement des informations, à l'aide de la commande **log_format** suivi d'une chaîne de caractères entre accolades. Vous pouvez inclure un certain nombre d'informations utiles, grâce aux raccourcis suivants :

```
%ruleid -> nom de la règle en vigueur
%sid    -> identifiant de l'appel système
%sname  -> son nom
%params -> ses paramètres
```

```
%pid    -> identifiant du processus appelant
%uid    -> identifiant réel de l'utilisateur du processus
%euid   -> identifiant effectif de l'utilisateur
%suid   -> sauvegarde de l'identifiant de l'utilisateur
%gid    -> identifiant du groupe de l'utilisateur
%egid   -> identifiant effectif du groupe
%sgid   -> sauvegarde de l'identifiant de groupe
%comm   -> nom de la commande responsable de l'appel système
%retval -> résultat renvoyé par l'appel système
           (uniquement dans le cas "when=after")
```

Dans notre exemple, pour chaque appel à *unlink*, une ligne supplémentaire apparaîtra dans le log, indiquant le nom de la commande, ses paramètres, ainsi que des informations sur l'utilisateur.

Le paramètre "**when**" peut prendre deux valeurs, "**before**" et "**after**". Quand on sait que la traduction littérale et respective des ces trois mots est "quand", "avant" et "après", on a tout compris... J'ai choisi d'effectuer l'action avant l'exécution de l'appel système.

Et voilà ! Nous allons pouvoir faire notre premier test. N'oublions pas l'accolade fermante ("}"), enregistrons le tout dans un fichier, disons /tmp/scttest.conf, que nous allons nous empresser de charger dans Syscalltracker.

sct_config upload /tmp/scttest.conf
Successfully uploaded rules from file '/tmp/scttest.conf'

Si la commande renvoie une ligne similaire, c'est que la règle est correctement chargée. Pour tester le résultat, ouvrez deux consoles (de préférence sous X, pour les avoir sous les yeux). Dans la première, vous observerez le log de Syscalltracker avec :

sctlog

Dans l'autre console, amusez vous à créer puis supprimer un fichier, par exemple :

```
# touch /tmp/dummy
# rm /tmp/dummy
```

Avec notre règle "**unlink_rule1**", vous verrez s'afficher dans le premier terminal :

```
"rm" : "/tmp/dummy" suppression par 0 --> 0
```

Un appel à *unlink* a eu lieu, provoqué par la commande "rm" avec comme paramètre "/tmp/dummy" et pour utilisateur effectif "root" (de fait le seul utilisateur à pouvoir supprimer quoi que ce soit vis–à–vis du système ;–)). Vous savez dorénavant écrire une règle pour Syscalltracker, et devriez être en mesure de décrypter la

suivante :

```
rule
{
   syscall_name = unlink
   rule_name = prevent_delete
   filter_expression {PARAMS[1]=="/etc/passwd" && UID == 0}
   action {
     type = FAIL
     error_code = -1
   }
   when = before
}
```

Il n'y a pas grande différence avec notre premier exemple. Nous définissons ici une action de type "**FAIL**" (échec) et retournons donc un code d'erreur, en l'occurrence "−1", ce qui signifie que l'opération n'est pas autorisée (voyez le contenu du fichier /usr/include/asm/errno.h pour connaître la liste des numéros d'erreur et leur sens).

La ligne commençant par "filter_expression" nous permet de limiter la porée de la règle : l'expression qui suit (entre les accolades) doît être vérifiée pour que la règle s'applique. Dans notre exemple, il est clair que je cherche à empêcher le super−utilisateur (root) de supprimer accidentellement le fichier /etc/passwd, et il est tout aussi clair que la méthode est loin d'être parfaite : considérez en effet les commandes suivantes, sans pour autant essayer !..

```
# cd /etc && rm -f ./passwd
```

ou encore

```
# rm -f /tmp/dummy /etc/passwd
```

Oublions ces faiblesses, qu'un administrateur chevronné saura aisément corriger : ce n'est après tout qu'un exemple ! J'insisterai par contre sur l'utilité d'appliquer l'action "**FAIL**" avant l'appel système, il serait en effet ridicule de vouloir signifier l'échec une opération qui vient d'être réalisée. Ajoutons maintenant cette règle dans le fichier de configuration puis rechargeons ce dernier :

```
# sct_config delete
# sct_config upload /tmp/try.conf
```

Attention ! Prenez bien garde de l'ordre dans lequel apparaissent vos règles, Syscalltracker appliquera la première vérifiée et n'ira pas plus loin. Il faut donc déclarer "prevent_delete" avant "unlink_rule1" si l'on veut éviter une catastrophe avec cette commande :

rm –f /etc/passwd

Avec "unlink_rule1" en tête, la règle étant vérifiée, "prevent_delete" sera tout bonnement ignorée, donc Syscalltracker se contentera de notifier dans le log la suppression d'un fichier pourtant vital, passant outre notre mesure de prévention... Si ces considérations vous dépassent, pensez bien à faire une copie de sauvegarde de /etc/passwd avant de faire le test ! Et pour ceux qui voudraient affiner la règle, le conseil est encore valable ;–)

Éléments d'analyse

Maintenant que vous connaissez les rouages de Syscalltracker, j'aimerais vous entretenir d'un appel système particulier, à savoir *ptrace()*. La page de manuel ("man ptrace") vous apprendra qu'il sert à observer et contrôler l'exécution d'un programme. C'est donc un outil aussi puissant que dangereux : s'il peut vous aider à traquer un bug, il permet aussi bien d'analyser des failles du système (et éventuellement les exploiter !). Pour garder une trace de telles activités, nous allons définir une nouvelle règle...

```
rule
{
    syscall_name=ptrace
    rule_name=ptrace_rule1
    action {
        type=LOG
      log_format {%comm : %params appel ptrace par %euid --> %suid}
  }
    when=before
}
```

...que nous nous empresserons de charger pour la tester. Pour ce faire, j'en profite pour introduire le programme *strace*, fourni avec la distribution Redhat. Au besoin, pour la Redhat 7.3 :

rpm –Uvh /chemin/vers/Redhat/RPM/strace–4.4–4.i386.rpm

Lancez maintenant "*sctlog*" dans une console (si ce n'est déjà fait), et dans une autre, par exemple :

strace /bin/ls

Une constatation s'impose à notre esprit : *strace* est un outil puissant pour observer les appels système provoqués par un fichier exécutable. L'observation du résultat de la commande précédente devrait suffire à vous en convaincre. Un autre effet de cette commande est, conformément à nos attentes, l'inscription dans le log d'un certain nombre de lignes de cet accabit :

```
"strace" : 3, 2019, 24, -1073748200 appel ptrace par 0 --> 0
"strace" : 24, 2019, 1, 0 appel ptrace par 0 --> 0
"strace" : 3, 2019, 44, -1073748200 appel ptrace par 0 --> 0
"strace" : 3, 2019, 24, -1073748200 appel ptrace par 0 --> 0
"strace" : 3, 2019, 0, -1073748216 appel ptrace par 0 --> 0
"strace" : 7, 2019, 1, 0 appel ptrace par 0 --> 0
```

De fait, la combinaison de strace et Syscalltracker constitue le couple idéal pour un audit système. Et vous en savez assez pour vous lancer dans l'aventure, Syscalltracker offrant ce qu'il faut de flexibilité pour parfaire votre maîtrise des appels système.

Dernières recommandations

Je n'oserais vous laisser voler de vos propres ailes sans mentionner auparavant ces quelques commandes utiles. Pour voir les règles qui sont chargées :

sct_config download

Pour une remise à zéro des règles :

sct_config delete

Et enfin, une fois que vous en avez fini avec Syscalltracker, autant libérer la mémoire.
Attention ! cette commande peut échouer, renvoyant ce message laconique : "Device or ressource busy". C'est que Syscalltracker est en cours d'activité, attendez un peu avant de ressayer.

sct_unload

En conclusion, j'ajouterai que Syscalltracker est sans danger pour le système et qu'il n'occasionne pas de charge excessive (à moins de charger une quantité outrageuse de règles :–)). Je vous invite donc vivement à définir vos propres règles et laissez Syscalltracker plonger à votre place dans les arcanes de votre système pour en dévoiler un à un les mystères...
