LinuxFocus article number 320
http://linuxfocus.org

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

Over de auteur:

Mijn naam is Mulyadi Santosa. Ik woon in Indonesië en werk als freelance auteur en consultant. Mijn interesses zijn clustering, systeembeveiliging en netwerken. Ook heb ik een klein bedrijf op cluster gebied, dat zich richt op cost-of-the-shelf clustering. OpenMosix en openSSI hebben echt mijn aandacht getrokken. In mijn vrije tijd zijn lezen en sport een welgekomen afleiding. Je kunt me bereiken op a_mulyadi@softhome om er via e-mail over te discussiëren.

Vertaald naar het Nederlands door:
Guus Snijders
<ghs(at)linuxfocus.org>

Linux Internals traceren met Syscalltracker

Kort:

Soms willen we ons Linux systeem nog nauwer in de gaten houden. Er zijn vele log-programma's, inbraak detectie tools, integriteits controle tools, en zo verder. Deze keer wil een een mechanisme introduceren om Linux op kernel niveau te monitoren, wat een betrouwbaarder en breder bereik levert.

Introductie

Op een dag zat ik op een mailing list, die een clustering middleware tool besprak. Een van de threads ging over een systeem anomaliteit die werd veroorzaakt door een kernel patch. Er reageerde een persoon die zei dat ze moesten proberen het probleem te reconstrueren, aan de hand van de stappen zoals door de eerste persoon gerapporteerd. Deze persoon gebruikte een tool genaamd Syscalltracker om hem te helpen te achterhalen waar het probleem door veroorzaakt werd. En ik vroeg me af "Wat voor tool is Syscalltracker? Wat zijn de mogelijkheden?" Voor een gewone gebruiker als ik, liet de naam "Syscalltracker" me achter met niks meer dan een mysterieus vermoeden :-)

Syscalltracker

(http://syscalltrack.sourceforge.net) is een verzameling kernel modules om te helpen system calls, intern gebruikt door de Linux kernel, te traceren. Waar dient het voor? We het gebruiken om de oorzaak van bepaalde systeem misdragingen te achterhalen die moeilijk zijn te vinden met gewone traceringen of debug mechanismen. Een eenvoudig voorbeeld: stel dat je een configuratie bestand hebt in de /etc directory, genaamd inetd.conf (basis configuratie voor de INTETd daemon). Deze had je geconfigureerd om verschillende systeem daemons te draaien en andere niet. Dan start je inetd en in de eerste instantie ging alles goed. Maar, plotseling verdwijnt /etc/inetd.conf. Gelukkig heb je backups gemaakt en snel haal je het bestand terug uit een recent backup. En dan herstart je inetd, gebaseerd op een nieuwe configuratie. Deze keer heeft iets of iemand newlines (lege regels) toegevoegd aan inetd.conf en een commando om mysterieuze daemons te starten. Je raakt verward... "wie deed dat?", "wordt het veroorzaakt door een daemon die door inetd zelf is gestart?", "Is er een exploit (zwakke plek) die probeert mijn systeem te compromitteren?". Snel "cat" je het systeem log, start je "top" en "ps" om te zoeken naar ongebruikelijke processen of gebruikers, maar er is niks te zien.

Er is een oplossing voor het achterhalen van dit soort problemen. Geen 100% perfecte oplossing, maar bruikbaar genoeg om belangrijke gevallen aan te pakken. Het is gebaseerd op het feit dat iedere actie of commando van een shell, user programma of daemon (met andere woorden, IEDER proces) een of meerdere systeem procedures moet uitvoeren, bekend onder de naam *system call*. Probeer je een bestand te verwijderen? Dat betekent dat je *unlink* aanroept. Je startte een shell script? Dan moet het *exec()* of *execve* hebben aangeroepen. Dus vrijwel iedere actie, gerelateerd aan het systeem, wordt direct geïntepreteerd als een system call. Dit is het basisidee waarom traceren op basis van system calls een geducht wapen kan zijn.

Interesse?

Dan kun je het gewoon proberen. In dit artikel gebruik ik RedHat Linux 7.3 als basis systeem. Richt je browser op http://syscalltrack.sourceforge.net en download het pakket van de download sectie. Voor dit artikel is syscalltrack-0.82.tar.gz gebruikt, zo'n 500 kB groot. Pak dit pakket uit in een directory, bijvoorbeeld /usr/src:

```
# tar xzvf syscalltrack-0.82.tar.gz
```

En controleer dat je de Linux kernel broncode in /usr/src hebt.

rpm -qa | grep -i kernel-source

of:

ls -al /usr/src/linux-2.4

Als een van hen een negatief resultaat oplevert, dien je deze te installeren. Hij is te vinden op de RedHat CD (#2):

rpm -replacepkgs -Uvh /pad/naar/jouw/RPM/kernel-source-2.4.18-3.i386.rpm

Let op dat je Syscalltracker tegen dezelfde versie van de broncode compileert als waar je Linux systeem mee draait. Bijvoorbeeld: als je een standaard RedHat 7.3 kernel gebruikt, dien je de kernel broncode van de Redhat CD te gebruiken. Of, als je je eigen Linux kernel gebruikt, moet je verderop Syscalltracker compileren tegen deze kernel broncode.

Behalve kernel broncode, heb je ook het kernel configuratiebestand van RedHat nodig, om het compileren te vereenvoudigen. Probeer de inhoud van /boot te bekijken:

ls -al config*

Als er een uitvoer is als ’conf ’config-2.4.18-3’, dan dien je deze te kopiëren naar /usr/src/linux-2.4. Hernoem dit bestand naar ’.config’.

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

Als dit bestand om een of andere reden ontbreekt, kun je het kopiëren van de kernel bron directory. Het bestand kan gevonden worden in de *configs* directory. Kies degene uit die overeenkomt met je draaiende kernel. Dit kun je achterhalen met

uname -a

Dit zou de versie van de kernel moeten geven. Je kunt het al raden. Stel dat de uitvoer "kernel-2.4.18-3-i386" bevat, dan dien je kernel-2.4.18-3-i386.config te gebruiken.

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

Nu kun je de volgende commando’s gebruiken

#cd /usr/src/linux-2.4.18-3
make mrproper
make menuconfig

Stel de juiste opties is en sla deze op. Als je een zelf gecompileerde kernel gebruikt maar je bent je oude

kernel config bestand kwijt dan dien je deze zorgvuldig te reconstrueren, om toekomstige problemen te voorkomen (ik hoop van niet :-)).

Syscalltracker compileren

Tot nog toe hebben we alle afhankelijkheden voorbereid. Nu kunnen we beginnen met de compilatie van Syscalltracker:

cd /usr/src/syscalltrack-0.82
./configure (of ./configure -with-linux=/pad/naar/jouw/linux/kernel/source)
make && make install

Als de compilatie met succes is afgesloten, zul je twee nieuwe modules vinden:

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

Dit zijn de modules die verantwoordelijk zijn voor jouw systeem tracering. De auteur gebruikt zelf de term *system call hijacking*, oftewel het onderscheppen van de system call en het voor-gescripte werk doen voordat de procedure wordt uitgevoerd. Vervolgens laadt je deze. Er is een script aanwezig om dat te doen

sct_load (as root)

Controleer of de kernel modules geladen zijn met *lsmod*. Je zou iets vergelijkbaars met het volgende moeten zien:

```
Module     Size  Used by    Not tainted
sct_rules  257788   2
sct_hijack 110176   1      [sct_rules]
<&hellip;..knip&hellip;&hellip;&hellip;..>
```

Regels

Gefeliciteerd! Je hebt de modules geladen en de Syscalltracker draait nu. Maar het is nog niet voorbij. Je moet nog *rules* (regels) aanmaken om Syscalltracker correct te laten werken. Laten we beginnen met een eenvoudige:

```
rule
 {
   syscall_name=unlink
   rule_name=unlink_rule1
   action
   {
     type=LOG
     log_format {%comm : %params delete by %euid --> %suid}
```

```
   }
   when=before
}
```

Iedere Syscalltracker rule begint met het *gereserveerde woord* "rule" (regel), gevolgd door een "{". Hierna geef je op welke system call je wilt observeren met de parameter "**syscall_name**". Er zijn vele system calls die je kunt gebruiken. Voor een complete lijst kun je kijken in het bestand /usr/local/lib/syscalltrack-0.82/syscalls.dat-2.4.18-3. De bedoeling van sommige system calls is soms lastig te raden, maar sommige zijn meer voor de hand liggend. Voorlopig gebruik ik *unlink*. Deze system call wordt gebruikt als iets of iemand probeert een bestand te verwijderen. Ik denk dat het een goede keuze is om mee te beginnen; het idee is dus om alle "verwijderingen" te monitoren die voorkomen op je systeem.

De naam van de regel geef je als parameter mee aan "**rule_name**". Deze is vrij te kiezen, gebruik gewoon een eenvoudig te begrijpen naam. Ik heb gekozen voor "**unlink_rule1**". Bij de sectie "**action**" defineer je de actie die Syscalltracker moet ondernemen wanneer hij een system call van dit type onderschept. Syscalltracker ondersteunt verschillende soorten acties; hier gebruiken we het **LOG** type. Deze actie schrijft een log boodschap naar **/dev/log**. Volgens de TODO lijst op de website zijn er plannen om system call rewriting (herschrijven) mogelijk te maken. Dit betekent dat je system call parameters kunt manipuleren en je eigen parameters kan injecteren :-)

Voor de **LOG** actie dien je een uitvoer formaat op te geven. Er zijn ingebouwde macros om gedetailleerde uitvoer te krijgen.

```
%ruleid -> regelnaam die overeenkomt met de onderschepte system call
%sid    -> system call identificatienummer
%sname  -> naam van de system call
%params -> system call parameter
%pid    -> proces ID dat de system call aanriep
%uid    -> user ID welke de system call aanriep
%euid   -> effectieve user ID die de system call aanriep
%suid   -> opgenomen user ID die de system call aanriep
%gid    -> het groep ID van de user die de system call aanriep
%egid   -> effective groep ID van de user die de system call aanriep
%sgid   -> opgenomen groep ID van de user die de system call aanriep
%comm   -> naam van het commando dat de system call aanriep
%retval -> retourwaarde van system call. Werkt alleen voor
           LOG actie met het type "after"
```

Voor dit voorbeeld schreef ik

.log_format {%comm : %params delete by %euid --> %suid}

Het betekent "ik wil ieder commando loggen dat de system call genaamd *unlink* uitvoert, met het effectieve user id en opgenomen user id"

In de parameter **when** kunnen we kiezen tussen "**before**" of "**after**". Het verschil moge duidelijk zijn, als we "**before**" gebruiken, zal het vastleggen geschieden voordat de system call wordt uitgevoerd. Als we "**after**" gebruiken, vindt het vastleggen na het uitvoeren van de system call plaats.

We sluiten de regel af met "}". Deze hele regel kan in een normaal tekst bestand worden geschreven, laten we het bijvoorbeeld "**try.conf**" noemen en opslaan in **/tmp**. Vervolgens laad je deze regel in

Syscalltracker:

sct_config upload /tmp/try.conf

Als de regel correct geschreven is, verschijnt de uitvoer "Successfully uploaded rules from file ’/tmp/try.conf’ ".

Ok, alles ging goed. Nu komt de testfase. Begin met een console, bijvoorbeeld een xterm in Xwindow. Op een console bekijk je Syscalltracker’s log met

sctlog

Spoedig zul je de uitvoer zien als resultaat van het onderscheppen van de system calls, als er een overeenkomt met je regels. Op een andere console, doe je iets als het volgende:

```
# cd /tmp
# touch ./dummy
# rm ./dummy
```

Met de bovengenoemde regel krijg je met sctlog waarschijnlijk een uitvoer als hieronder

```
"rm" : "./dummy" delete by 0 --> 0
```

Deze uitvoer vertelt je dus exact wat er gebeurd is:

Het commando "**rm**" met de parameter "**./dummy**" voert de system call *unlink* uit. Met andere woorden: rm werd gebruikt om een bestand te verwijderen. Dit commando draait onder een effectief user id gelijk aan 0 (root dus).

Hier is een andere voorbeeld regel:

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

Dit is vergelijkbaar met ons eerste voorbeeld, maar nu gebruiken we de **FAIL** actie. Speciaal voor **FAIL** moeten we de retour waarde voor de onderschepte system call definiëren. Hier gebruik ik "-1" of "operatie niet toegestaan". De complete lijst met nummers kan gevonden worden in /usr/include/asm/errno.h.

Op de regel met "filter expression" geef je een voorwaarde aan waarop wordt gecontroleerd: als de eerste parameter (of system call) overeenkomt met "**/etc/passwd**". Hiervoor hebben we de **PARAMS** variabele nodig. Opmerking: iedere sectie heeft zijn eigen verplichte parameters. Let ook op dat deze controle niet perfect is, omdat er ook iets als "cd /etc && rm -f ./passwd" gebruikt kan worden. Maar dit

is OK om mee te beginnen. We kunnen bijvoorbeeld ook controleren of het UID gelijk is aan 0 (root).

Voeg deze regel toe aan je vorige rule-bestand en laad opnieuw:

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

Merk op dat de volgorde van de regels belangrijk is. Als je de regel "prevent_delete" (voorkom verwijderen) vóór "unlink_rule1" opneemt, dan kun je het volgende doen:

rm -f /etc/passwd

De system call zal als eerst regel "prevent_delete" tegenkomen, en de hele actie zal falen. De "unlink_rule1" die later komt wordt genegeerd. Maar als je de volgorde omdraait ("unlink_rule1" voor "prevent_delete") dan krijg je alleen het log, zonder de actie te stoppen!

Er is nog een andere system call die interessant is om te bekijken. Deze is genaamd *ptrace*. Uit "man ptrace" kun je leren dat deze system call wordt gebruikt om het uitvoeren van een ander programma te controleren. In de juiste handen kan deze *ptrace* een handige debugging tool zijn, in de verkeerde handen kan het worden gebruikt voor het analyseren en misbruiken van beveiligingsgaten. Laten we dus een regel toevoegen om deze te loggen.

Om dit te doen, voeg je een regel als dit toe:

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

Merk op dat we ’ptrace’ opgeven als system call om te observeren. Om dit te testen kun je het programma *strace* gebruiken. Laad eerst de bovenstaande regel in Syscalltracker en start sctlog. Dan start je strace tegen bijvoorbeeld *ls*:

strace /bin/ls

In sctlog, krijg je meerdere regels als deze:

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

Voor de mensen die niet bekend zijn met strace, dit is een (krachtige) tool om system calls te achterhalen, zoals aangeroepen in een uitvoerbaar bestand. Strace gebruikt intern ptrace om zichzelf aan het doel programma te hangen, zodat deze getraceerd kan worden. Eigenlijk zijn strace en Syscalltracker

een ideale combo voor systeem- en bestand auditing, dus ik denk dat het de moeite waard is ze te vermelden. Redhat 7.3/8/9 beschikt er al over. Je hoeft alleen de RPM te installeren (hier Redhat 7.3):

rpm -Uvh /pad/naar/je/Redhat/RPM/strace-4.4-4.i386.rpm

Nu ben je een stap verder in de diagnose van je systeem. Syscalltracker geeft de gebruiker flexibiliteit en de mogelijkheid om bij te leren over system calls. Ter aanvulling: als je de regels wilt bekijken die geladen zijn, gebruik je:

sct_config download

Om alle geladen regels te verwijderen, typ:

sct_config delete

Ten slotte, als je Syscalltracker niet meer nodig hebt, kun je deze uit het geheugen verwijderen:

sct_unload

Het is mogelijk dat dit commando er in faalt om Syscalltracker te verwijderen met de waarschuwing "Device or resource busy". Als dit gebeurt, is het mogelijk dat Syscalltracker bezig is. Laat het een tijdje werken en probeer opnieuw. Syscalltracker is veilig voor het systeem en zal het systeem niet zwaar belasten (tenzij je tonnen regels toevoegt ;)). Conclusie: Laat Syscalltracker in je kernel zitten en laat het zijn werk doen. Met voldoende regels kun je beginnen je systeem van dichtbij te observeren.

Site onderhouden door het LinuxFocus editors team © Mulyadi Santosa "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Vertaling info: de --> -- : Mulyadi Santosa <a_mulyadi/at/softhome.net> en --> nl: Guus Snijders <ghs(at)linuxfocus.org>

2005-01-14, generated by lfparser_pdf version 2.51