Systemaufrufe mit Systrace steuern

Stefan Schumacher
Stefan.Schumacher@Kaishakunin.com

KAISHAKUNIN.COM
IT-Sicherheitsberatung

Veröffentlicht in der GUUG UpTimes Dezember 2007.

Literaturverzeichnis

```
@article{Schumacher-ut-systrace,
  author = {Stefan Schumacher},
  title = {Systemaufrufe mit Systrace steuern},
  year = {2007},
  month = "12",
  volume="4",
  publisher = {German Unix User Group},
  journaltitle="UpTimes",
  pages="12\,--\,19",
  issn="1860-7683",
  language = {german},
  url={http://kaishakunin.com/publ/guug-uptimes-systrace.pdf},
  urldate="2009-11-07",
}
```

Abbildung 4: Einblick in die kleinere T5120 (rechts in der Abbildung ist hinten)

Systemaufrufe mit Systrace steuern

Stefan Schumacher <stefan@net-tex.de>

Systrace ermöglicht unter NetBSD, OpenBSD, FreeBSD, Linux und MacOS X die Überwachung und Steuerung von Systemaufrufen. Dazu benutzt es Richtlinien, die für jedes verwendete Programm definiert werden. Anhand dieser erlaubt oder verbietet Systrace Systemaufrufe oder führt sie unter anderen Benutzerrechten aus. Somit kamm man SETUID-Programme als unpriviliegierter Benutzer sowie nur bestimmte Systemaufrufe mit Root-Rechten ausführen. Systrace erlaubt daher die Implementierung einer feingranulierten Sicherheitsrichtlinie, die sogar Argumente von Systemaufrufen überprüfen kann.
Dieser Artikel beschreibt die Funktionsweise von Systrace, Aufbau und Erzeugung einer Richtlinie sowie den praktischen Einsatz anhand zweier Beispiele unter NetBSD.

Wie allgemein bekannt, ist Verteidigung eines Rechnersystems erfolgreicher beim Vorhandensein mehrerer Verteidigungslinien. Diese Linien müssen sich überlappen, ohne jedoch voneinander abzuhängen. Dies können beispielsweise Paketfilter, Application-Level-Gateways oder biometrische Zugangskontrollen sein. Allerdings bietet jede Verteidigungslinie potenziell wiederum neue Angriffspunkte. So könnte ein Eindringling beispielsweise ein Einbruchserkennungssystem (IDS) durch einen Speicherüberlauf übernehmen und mit Root-Rechten unter seine Gelwalt bekommen. Daher ist es grundsätzlich angeraten, Schadmöglichkeiten von Programmen einzugrenzen.

Fast jedes heute eingesetzte Programm ist zu komplex und zu umfangreich, um es manuell vollständig auf Fehlerquellen und Sicherheitslücken hin zu überprüfen. Selbst wenn es als quelloffenes Programm vorliegt, findet eine Überprüfung auf absichtliche oder fahrlässige Sicherheitsprobleme zu selten statt. Quelloffenheit ist zwar ein sehr gutes Kriterium für sicherheitsrelevante Programme, schützt

aber allein definitiv nicht vor Lücken oder gar Hintertüren, wie in [2] zu lesen ist. Es ist nicht möglich, den Schluss „quelloffene Software bedeutet automatisch Freiheit von Sicherheitslücken“ zu ziehen. Stellenweise ist man auf den Einsatz geschlossener Software angewiesen, hat dann also keinerlei Möglichkeit mehr, den Quellcode zu prüfen und muss ihr zwangsläufig trauen.

Angriffe gegen Systeme konzentrieren sich in der Regel auf Systemaufrufe (*system calls*, abgekürzt *Syscalls* genannt), die unter anderem dazu dienen, um im Kernel privilegierte Operationen durchzuführen. Um derartigen Angriffen zu begegnen, ist ein System nützlich, das die Ausführung von Systemaufrufen begrenzt und die Zugriffsrechte darauf feiner als bisher granuliert. Hierzu benötigt man eine Richtlinie, die Zugriffe reglementiert. Diese Richtlinie muss alle nur möglichen Fälle abdecken sowie Kenntnisse aller auftretenden Pfade haben, was Dank symbolischer Links nicht besonders einfach ist. Die Richtlinie beschreibt das Normalverhalten eines Prozesses und dient so als Vergleich zum laufenden System. Weicht ein Prozess von der beschriebenen Richtlinie ab, wird dies als Einbruchsversuch erkannt und verhindert. Außerdem soll das überwachende Programm Protokolle der überwachten Programme erzeugen, um so Anomalien erkennen und analysieren zu können.

Ein derartiges System lässt sich auf verschiedene Weise implementieren. Einerseits kann es komplett im Kernel arbeiten, ebenso komplett im User-Space. Im Kernel ist die Ausführung recht schnell, aber dies ist äußerst komplex und nur schwer auf andere Betriebssysteme zu portieren. Im User-Space hingegen ist solch ein Programm portabel, aber langsam und unsicherer, da es zu Race-Conditions zwischen dem Zeitpunkt der Analyse und der Ausführung eines Systemaufrufes kommen kann.

Kluger Kompromiss

Systrace, von seinem Erfinder Niels Provos in [1] beschrieben, verwendet einen hybriden Ansatz. Es wird ein kleiner Teil im Kernel implementiert und der größte Teil im User-Space. Der Kernel-Teil ermöglicht die schnelle Behandlung von kontext*insensitiven* Systemaufrufen, die beispielsweise stets abgelehnt oder erlaubt werden. Weiterhin ermöglicht dies, auszuführende Programme in einem Sandkasten zu kapseln und geforkte Prozesse mit der vererbten Richtlinie weiter zu kontrollieren. Der Teil im User-Space überwacht die Systemaufrufe auf kontext*sensitive* Entscheidungen und trifft sie anhand der definierten Richtlinie. Während der Entscheidungsfindung blockiert der Kernel den fraglichen Prozess. Weiterhin kann der User-Space-Dämon über eine Schnittstelle Informationen wie Zustandsübergänge, PID-Änderungen oder Forks vom Kernel-Teil anfordern.

Mittels Systemaufrufe können Anwenderprogramme Kernelfunktionen nutzen. So kann sich ein Programm etwas an einen niedrigen Port binden oder eine System-Logdatei öffnen, da solche Operationen System-Rechte erfordern. Damit ein Programm solch privilegierte Systemaufrufe tätigen darf, muss man es mit Root-Rechten starten – und eröffnet damit eine potenziell riesige Sicherheitslücke. Systrace von Niels Provos löst dies, indem es die Rechtezuteilung nicht mehr auf Programmebene vornimmt, sondern auf Ebene der Systemaufrufe herunterbricht. Ein von einem normalen Benutzer gestartetes Programm bekommt gemäß einer vorher erstellten Richtlinie von Systrace entsprechende Zugriffsrechte auf Systemaufrufe zugeteilt.

Die Grammatik der Richtlinie

Zur Definition der Richtlinie für ein zu überwachendes Programm dient eine recht einfache Grammatik mit aneinander gereihten Regeln. Eine Regel besteht dabei aus einem booleschen Ausdruck und der auszuführenden Aktion. Die Aktion kann einer der folgenden sein:

- *ask* (frage),
- *deny* (verbiete) oder
- *permit* (erlaube)

in Verbindung mit optionalen Argumenten. Ergibt der boolesche Ausdruck *wahr*, wird die definierte Aktion ausgeführt. Bei der Aktion *ask*, erfolgt eine Rückfrage beim Benutzer, um die Aktion zu erlauben oder zu verbieten.

Der boolesche Ausdruck setzt sich aus verschiedenen Variablen und den Logik-Operatoren *and* (logisches Und), *or* (logisches Oder) und *not* (logisches Nicht) zusammen. Die Variablen bestehen aus den normalisierten Systemaufruf-Namen, den dem Systemaufruf übergebenen Argumenten und einem Logik-Operator, der beide Argumente verknüpft.

Die Filterausdrücke auf Argumente verwenden verschiedene Operatoren:

- *match* Wahr, wenn der Dateiname gemäß den Regeln in `fnmatch(3)` übereinstimmt.
- *eq* Wahr, wenn das Argument des Systemaufrufes genau der Vorgabe entspricht.
- *neq* Die logische Negation des *eq*-Operators.
- *sub* Wahr, wenn die angegebene Teilzeichenkette im Argument des Systemaufrufes vorkommt.
- *nsub* Die logische Negation des *sub*-Operators.
- *inpath* Wahr, wenn das Argument des Systemaufrufes im Pfad der Vorgabe vorkommt.
- *re* Wahr, wenn das Argument des Systemaufrufes dem angegebenen regulären Ausdruck entspricht.

Zur Veranschaulichung, was Systrace bietet, folgen hier ein paar Ausdrücke:

- `netbsd-execve: permit` Erlaubt alle `execve(2)`-Aufrufe.
- `netbsd-execve: true then permit log` Erlaubt alle `execve(2)`-Aufrufe und protokolliert sie an Syslog. `true then` ist nötig für den Einsatz von `log`.
- `netbsd-fsread: filename eq ''/etc/passwd'' then permit` Erlaubt alle Lese-Operationen auf `/etc/passwd`.
- `netbsd-fsread: filename match ''/etc*'' then deny [eaccess] log` Verbietet alle Lese-Operationen auf `/etc*` mit dem Fehlercode EACCESS und protokolliert sie an Syslog.
- `netbsd-seteuid: uid eq ''1007'' or uname eq ''systraced'' then permit` Erlaubt das Setzen der effektiven UID, wenn die UID des aufrufenden Benutzers 1007 ist oder er der Benutzer *systraced* ist.

- `netbsd-connect: sockaddr re ''inet-.192\.168\.[0,4,8]\.[4-6]. :22'' then permit log` Erlaubt eine Socket-Verbindung, wenn die Zieladresse des Sockets im angegebenen Adressbereich liegt. Es folgt wie zuvor eine Protokollierung des Aufrufs.

Um über einen Systemaufruf zu entscheiden, traversiert Systrace alle Ausdrücke und bricht beim ersten Ausdruck ab, der zum Systemaufruf passt. Dieser Ausdruck entscheidet dann, ob der Systemaufruf ausgeführt oder abgelehnt wird. Bei keinem passenden Ausdruck delegiert Systrace die Entscheidung an den Benutzer. Bei abgelehntem Systemaufruf kann Systrace an das aufrufende Programm einen spezifizierten Fehlercode zurückgeben.

Um die Richtlinie auf Benutzer- bzw. Gruppenebene granulieren zu können, kann man Ausdrücke mit einem Prädikat versehen. Dieses Prädikat genügt der Form *", if"* {*"user"*, *"group"*} {*"="*, *"!="*, *"<"*, *">"* } {*Bernutzername, numerische UID*}. Somit lassen sich Ausdrücke der Art `netbsd-fsread: filename eq ''/etc/master.passwd'' then deny[eperm], if group != wheel` erzeugen. Hier lehnt Systrace den Zugriff auf die Datei `/etc/master.passwd` mit dem Fehlercode `EPERM` ab, wenn der aufrufende Benutzer nicht Mitglied der Gruppe *wheel* ist.

Will man ein Systemaufruf unter anderen Benutzerrechten ausführen, kann man an den Ausdruck die Direktive *as user:group* angehängen. Beispiele dazu stehen weiter unten im Kapitel *Apache überwachen*.

Durch das Anhängen der Log-Direktive log an den Ausdruck einer Richtlinie erreicht man, dass der Ausdruck selber sowie die durch ihn implizierte Entscheidung vom Betriebssystem ins System-Log kommt. Mit dieser Option lassen sich Programme komplett überwachen und analysieren. Protokolliert man beispielsweise alle `exec(3)`-, `execve(2)`- und `connect(2)`-Aufrufe, erfährt man, welche Programme ein Benutzer ausgeführt und welche Sockets er geöffnet hat. Beachten Sie hierbei aber unbedingt datenschutzrechtliche Bestimmungen und andere Regelungen.

Erzeugung einer Richtlinie

Ziel der Richtlinie ist es, alle erlaubten Systemaufrufe einer Anwendung zu erfassen. Nicht erfasste Aufrufe werden als Angriff gewertet und verboten. Somit lässt sich eine Richtlinie erstellen, in dem ein spezielles Programm die zu erfassende Anwendung bei einem Probelauf überwacht und die abgesetzten Systemaufrufe mitschneidet. Die ausgeführten Systemaufrufe werden dabei in die Systrace-eigene kanonische Form normalisiert und in Richtlinienausdrücke übersetzt. Existiert in der bisherigen Richtlinie kein Ausdruck, der den aktuellen Systemaufruf behandelt, wird ein neuer Ausdruck angehängt, der den ausgeführten Aufruf erlaubt. Manuelles Eingreifen in die erzeugte Richtlinie ist in der Regel nicht erforderlich, es sei denn, die überwachte Anwendung arbeitet mit Zufallskomponenten, wie bespielsweise Zufallsnamen für temporäre Dateien. In diesem Fall muss man den entsprechenden Ausdruck dahingehend abändern, das Systrace die Zufallskomponente im Dateinamen (wie eine Prozess-ID) verarbeiten kann. Bei dieser automatisierten Richtlinienerstellung impliziert man, dass das zu überwachende Programm per se sicher ist. Kann man dies nicht gewährleisten, ist eine so erstellte Richtlinie nicht als sicher zu betrachten. Außerdem ist die Automatik ebenfalls nicht anwendbar, wenn es nicht möglich ist, alle auftretenden Code-Pfade durchzuexerzieren. Durchsucht nämlich ein Programm bei jedem Aufruf andere Pfade oder Dateien, müssen alle möglichen Pfade erfasst werden. Dies ist mit einem simplen Durchlauf von Systrace nicht möglich. Stattdessen muss der Administrator alle möglichen Pfade in der Richtlinie angeben. Dies erledigt er meist mittels regulärer Ausdrücke.

Trotzdem dient eine so erstellte Richtlinie als Basis für eine händische Anpassung, oder als Basis für weitere Übungsläufe. Bei diesen wird die Richtlinie nur dann erweitert, wenn neue Systemaufrufe erfolgen. Hier kann der Benutzer wieder die resultierende Aktion festlegen. Nach dem Fertigstellen einer Richtlinie fertiggestellt, kann sie Systrace gegenüber dem gewünschten Programm durchsetzen.

Implementierung

Systrace kann in einem von drei verschiedenen Modi laufen:

- Initialisierungsmodus: Systrace überwacht ein Programm automatisch und generiert eine Richtlinie. Diese ist ein guter Startpunkt um eine angepasste und verfeinerte Richtlinie zu erzeugen.
- Nachfragemodus: Hier überwacht Systrace ebenfalls ein Programm und erzeugt eine Richtlinie, allerdings muss der Benutzer jedem Systemaufruf explizit zustimmen. Dies ist sinnvoll, wenn man dem zu überwachenden Programm nicht unbedingt vertrauen kann. Die Nachfrage erfolgt über das X-Programm `xsystrace(1)` oder im Textmodus ohne X.
- Überwachungsmodus: Systrace überwacht ein Programm und setzt die definierte Richtlinie durch. Es lehnt nicht erlaubte Systemaufrufe ab und protokolliert diese.

Systrace verfügt über eine Reihe von Optionen:

- `-A`: Initialisierungsmodus, erzeugt eine Richtlinie, in der alle Systemaufrufe erlaubt sind.
- `-a`: Überwachungsmodus, setzt die definierte Richtline durch.
- `-c UID:GID`: Spezifiziert eine numerische Benutzer- und Gruppen-ID, unter der das zu überwachende Programm ausgeführt wird. Nur als Root machbar.
- `-d Verzeichnis`: Setzt ein Verzeichnis für die Richtliniendateien. Standard ist `/.systrace`.
- `-f Datei`: Systrace verwendet die Richtlinie, die in der Datei angegeben ist.
- `-g gui`: Aktiviert einfachen graphischen Dialog für Systrace (s. Abb. 4)
- `-i`: Vererbt die Richtlinie des Elternprozesses an die Kinder.
- `-p`: Systrace bindet sich an den bereits laufenden Prozess mit der angegebenen PID. Der komplette Programmpfad muß ebenfalls angegeben werden.
- `-t`: Textmodus, läuft auch ohne X.
- `-U`: Benutzt nur globale Richtlinien (`/etc/systrace`) statt lokaler.
- `-u`: Deaktiviert das Zusammenfassen von Systemaufrufen zu Aliasen.

Sicherheit des Systems

Ein System wie Systrace hat selbst auch mit einigen Sicherheitsproblemen zu kämpfen wie Aliase auf Systemressourcen, Dateinamen von Programmen, die in einer Chroot-Umgebung laufen oder der Verfolgung von Prozess-IDs.

Ein Zugriff auf ein und dieselbe Datei ist unter Unix dank symbolischer Links und relativer Pfadnamen auf unendlich viele Arten denkbar. Außerdem können Netzdienste wie Proxy, NFS oder CFS, Dateien zur Verfügung stellen. Derartige Dienste sind für Systrace nicht sichtbar, der sollte aber trotzdem korrekt funktionieren.

Weiterhin ist es möglich, eine Race-Condition zu produzieren, die eine Systrace-Sandkiste aushebelt. Systrace benötigt für die Überprüfung eines Systemaufrufes eine gewisse Zeitspanne. Während dieser Zeitspanne kann ein anderer Prozess den eigentlichen Systemaufruf des zu überwachenden Programmes ändern. Systrace erkennt diese Änderung nicht und gestattet die Ausführung des inzwischen geänderten Systemaufrufes.

Um diesen Problemen zu begegnen, verwendet Systrace nur normalisierte Dateinamen und Systemaufrufe. Alle Dateinamen und Parameter für Systemaufrufe normalisiert Systrace, indem es Dateinamen in eine absolute Form – ohne Symlinks oder relative Pfadangaben – umgewandelt. Diese normalisierten Werte übergibt Systrace wieder dem Betriebssystem. Ausgenommen hiervon sind nur einige Systemaufrufe, wie `readlink`. Weiterhin sind diese normalisierten Werte in einem nur-lesbaren Puffer zwischengespeichert, so dass kein anderer Prozess die Werte manipulieren kann. Der Kernel verweigert die Ausführung von Systemaufrufen, die Symlinks als Argumente enthalten. Somit sind nur noch von Systrace normalisierte Aufrufe gestattet. Alle anderen Aufrufe lehnt Systrace ab.

Bei abgelehnten Systemaufrufen müssen die überwachten Programme eine entsprechende Fehlermeldungen bekommen. Da leider nicht alle Programme eine funktionierende Fehlerbehandlung implementieren, kann man in der Richtlinie ein bestimmter Fehlercode spezifizieren.

Ebenfalls zu betrachten ist ein Richtlinienwechsel und Prozessbeendigung. Wenn ein überwachter Prozess einen neuen Prozess startet, entfernt der Kernel den alten Prozess aus dem Speicher und führt stattdessen den neuen Prozess aus. Dieser neue Prozess kann ein vertrauenswürdiges Programm sein, so dass keine weitere Überwachung notwendig ist. Oder es kann ein Prozess sein, den Systrace mit einer anderen Richtlinie besser überwacht.

Der `log`-Befehl protokolliert in einer Richtlinie jeden ausgeführten Systemaufruf. Somit ist es möglich, mit einer Systrace-überwachten Shell alle `execve`-Aufrufe eines Benutzers zu überwachen. Wie oben schon erwähnt, sind dabei unbedingt datenschutz- und andere rechtliche Regelungen zu beachten.

Alle Richtlinien sind in einzelnen Dateien gespeichert. Kann ein Einbrecher die Richtlinien-Dateien manipulieren, kann er Systrace aushebeln. Daher sind die Richtlinien-Dateien unbedingt zu schützen. Dazu kann man neben restriktiven Schreibrechten die NetBSD-Dateiflags (`schg`) in Verbindung mit den Security-Level verwenden. Möchte man diese Methode nicht einsetzen, sollte man die Richtlinien zumindest regelmäßig mit einem Integritätsprüfer wie `mtree(8)`, AIDE oder Tripwire überprüfen.

Apache überwachen

Abbildung 1 zeigt den Start von Apache unter der Kontrolle von Systrace. Dies hat die Erstellung einer Richtlinie zur Folge, die nach Beendigung von Apache wieder weggeschrieben wird.

Ein Teil der Richtlinie ist in Abbildung 2 gezeigt.

In den Zeilen 11, 13 und 37 sieht man einen Zugriff auf eine Datei im `/var/run`-Verzeichnis. Wie man leicht erkennt, enthält der Dateiname die Prozess-ID bzw. Semaphorennummer, die man später durch den Joker *≪ ersetzen sollte. Gleiches gilt für den Operator eq≪ der durch match≪ auszutauschen ist, wie in Abbildung 3 gezeigt.

In den Zeilen 31 und 32 (Abbildung 2) bindet sich Apache an die angegebenen IP-Adressen auf Port 80. In den restlichen sowie ausgelassenen Zeilen liest Apache diverse Konfigurationsdateien ein. Startet man nun Apache unter Systrace-Überwachung, kann kein Benutzer auf die Webseiten zugreifen, da die Richtlinie keinen Lesezugriff für `/home/www` beinhaltet. Die Verstöße sind `/var/log/messages` zu entnehmen. Komfortabel kann man das beheben, in dem man Apache wieder im Initialisierungsmodus von Systrace startet: `systrace apachectl start`. Beim Aufruf von Adresse http://127.0.0.1/ mittels Browser meldet sich `xsystrace` – ein Beispiel findet sich in Abbildung 4 – zu Wort und verlangt vom Benutzer die Bestätigung oder Verweigerung der benötigten Systemaufrufe.

Um dies nicht zum Geduldsspiel ausarten zu lassen – diese Prozedur verlangt für jede einzelne vom Apache geladene Datei eine Aktion – bricht man vorzugsweise nach dem Laden der Indexseite ab und beendet Apache mit `apachectl stop`. Systrace hat die Richtlinie nun um die Lesezugriffe auf `/home/www` erweitert – allerdings für jede Datei einzeln, so dass man hier wieder reguläre Ausdrücke mit match≪ und *≪ eingesetzen sollten. Abbildung 5 zeigt die automatisch generierte Richtlinie, die jede aufgerufene Datei einzeln behandelt.

Abbildung 6 zeigt eine Verfeinerung der Richtlinie mit regulären Ausdrücken, so dass Zugriffe auf `/home/www/public/*` gestattet und auf `/home/www/intern/*` verboten sind. Weiterhin sind Zugriffe auf CGI-Dateien erlaubt, respektive verboten. Letzteres mit ENOENT, so dass für Apache die Dateien nicht existieren. Bis jetzt ist die Richtlinie soweit konfiguriert, das Apache unter Überwachung normal funktioniert.

Trotzdem muss Apache noch initial als Root gestartet werden (privilegierter Port 80). Systrace kann mit der Option `-c` zu überwachende Prozesse unter beliebigen numerische Benutzer- und Gruppen-ID starten. Dazu erfordert es allerdings noch die Anpassung der Richtlinie, da ein nichtprivilegierter Benutzer keine privilegierten Operationen ausführen darf. Systrace kann jede Aktion in der Richtlinie als ein anderer Benutzer ausführen. Anwendungen kann man so unter einer „normalen“ Benutzer-ID starten, nur die benötigten Systemaufrufe laufen dann unter Root. Dies erfordert jedoch den Start von Systrace nebst dem Programm unter Root. Startet man Apache nun

```
# systrace -A apachectl start
/usr/pkg/sbin/apachectl start: httpd started
# apachectl stop
/usr/pkg/sbin/apachectl stop: httpd stopped
# ls /root/.systrace/
usr_pkg_sbin_apachectl usr_pkg_sbin_httpd
#
```

Abbildung 1: Automatisch eine Richtline für Apache erstellen

```
[...]
        netbsd-pread: permit
        netbsd-fsread: filename eq "/etc/group" then permit
        netbsd-fsread: filename eq "/usr/pkg/etc/httpd/httpd.conf" then permit
        netbsd-fsread: filename eq "/usr/pkg" then permit
        netbsd-fsread: filename eq "/usr/pkg/etc/httpd/srm.conf" then permit
        netbsd-fsread: filename eq "/usr/pkg/etc/httpd/access.conf" then permit
        netbsd-gettimeofday: permit
        netbsd-fsread: filename eq "/etc/etc.network/resolv.conf" then permit
        netbsd-fsread: filename eq "/etc/hosts" then permit
        netbsd-chmod: filename eq "/var/run/httpd.mm.2872.sem"
                                  and mode eq "600" then permit
        netbsd-chown: filename eq "/var/run/httpd.mm.2872.sem"
                                  and uid eq "1002" and gid eq "-1" then permit
        netbsd-fswrite: filename eq "/var/log/httpd/error_log" then permit
        netbsd-dup2: permit
        netbsd-select: permit
        netbsd-fsread: filename eq "/usr/pkg/etc/httpd/mime.types" then permit
        netbsd-fsread: filename eq "/usr/pkg/etc/httpd/magic" then permit
        netbsd-fswrite: filename eq "/var/log/httpd/access_log" then permit
        netbsd-chdir: filename eq "/" then permit
        netbsd-fork: permit
        netbsd-exit: permit
        netbsd-setsid: permit
        netbsd-fsread: filename eq "/dev/null" then permit
        netbsd-fswrite: filename eq "/dev/null" then permit
        netbsd-socket: sockdom eq "AF_INET" and socktype eq "SOCK_STREAM" then permit
        netbsd-setsockopt: permit
        netbsd-bind: sockaddr eq "inet-[0.0.0.0]:80" then permit
        netbsd-listen: permit
        netbsd-bind: sockaddr eq "inet-[192.168.0.5]:80" then permit
        netbsd-bind: sockaddr eq "inet-[127.0.0.1]:80" then permit
        netbsd-fsread: filename eq "/var/run/httpd.pid" then permit
        netbsd-umask: permit
        netbsd-fswrite: filename eq "/var/run/httpd.pid" then permit
        netbsd-write: permit
        netbsd-fswrite: filename eq "/var/run/httpd.lock.1827" then permit
```

Abbildung 2: Automatisch erzeugte Richtlinie für Apache (Auszug)

mit `systrace -c 1002:1001 apachectl start` als Benutzer und Gruppe www≪ mit der bisherigen Richtlinie, wird sich Systrace erneut zu Wort melden mit den Systemaufrufen, die der Benutzer www≪ nicht ausgeführen durfte. Am einfachsten editiert man daher vorher die Richtlinie mit `vi(1)` und sucht zuerst nach Schreibzugriffen mittels `netbsd-fswrite`. Das umfaßt in diesem Beispiel die Log-, PID- und Semaphoren-Dateien. Die nächsten offensichtlichen Kandidaten sind alle `netbsd-bind`-Aktionen, in denen sich Apache an die Netzwerkschnittstelle bindet. Die angepassten Aktionen der Richtlinie für nichtprivilegierte Läufe von Apache unter Systrace als Benutzer www≪ finden Sie in Abbildung 7.

Systrace-überwachte Shell

Matthias Petermann beschreibt in [3] die Einrichtung einer Shell zur kompletten Überwachungen von Systrace.

Um dies zu ermöglichen, müsste der Start der Shell über Systrace in der Art `systrace -a -i ksh` erfolgen. Da solch ein Befehl nicht

```
[...]
        netbsd-chmod: filename match "/var/run/httpd.mm.*.sem"
                                    and mode eq "600" then permit
        netbsd-chown: filename match "/var/run/httpd.mm.*.sem"
                                    and uid eq "1002" and gid eq "-1" then permit
        netbsd-fswrite: filename match "/var/log/httpd/error_log" then permit
[...]
        netbsd-fswrite: filename match "/var/run/httpd.lock.*" then permit
```

Abbildung 3: Automatisch erzeugte Richtlinie korrigiert für Apache (Auszug)

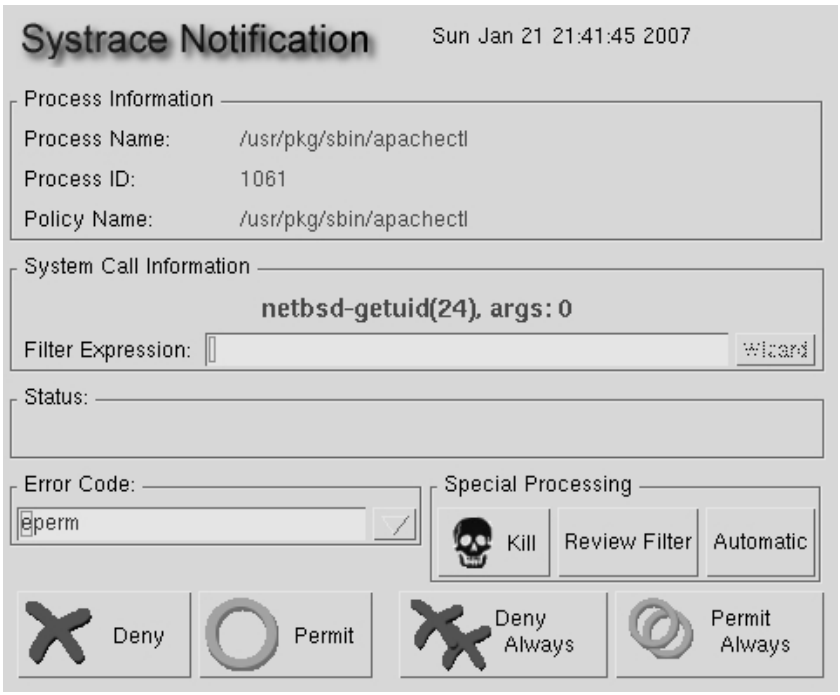

Abbildung 4: Xsystrace bittet den Benutzer um eine Entscheidung

```
[...]
        netbsd-fsread: filename eq "/home/www" then permit
        netbsd-fsread: filename eq "/home/www/.htaccess" then permit
        netbsd-fsread: filename eq "/home/www/index.html" then permit
        netbsd-fsread: filename eq "/home/www/index.pl" then permit
        netbsd-fsread: filename eq "/home/www/index.mhtml" then permit
[...]
```

Abbildung 5: Automatisch generierte Richtlinie für Apache, die jede Datei einzeln aufführt.

```
[...]
        netbsd-fsread: filename match "/home/www/public/*" then permit
        netbsd-fsread: filename match "/home/www/intern/*"
                                        then deny [enoent]
        netbsd-fsread: filename eq "/home/www/cgi-bin/cvsweb.cgi" then permit
        netbsd-fsread: filename eq "/home/www/cgi-bin/postgresql.cgi"
                                        then deny [enoent]
[...]
```

Abbildung 6: Angepasste Apache-Richtline

in der `/etc/master.passwd` stehen darf, dient ein kleines C-Programm (Abbildung 8) zur Kapselung des Befehls. Diese Kapsel muss nun nach Erstellung der Richtlinie in `/etc/shells` und `/etc/master.passwd` für die jeweils zu überwachenden Benutzer als Shell eingetragen werden. In diesem Beispiel soll der Benutzer systraced« eine eingeschränkte Korn-Shell erhalten.

```
# grep 'as root' usr_pkg_sbin_httpd
  netbsd-fswrite: filename match "/var/run/httpd.mm.*.sem" then permit as root
  netbsd-fswrite: filename match "/var/run/httpd.mm.*.sem" then permit as root
  netbsd-fswrite: filename match "/var/run/httpd.mm.*.sem" then permit as root
  netbsd-chmod: filename match "/var/run/httpd.mm.*.sem" and mode eq "600"
                                                         then permit as root
  netbsd-chown: filename match "/var/run/httpd.mm.*.sem" and uid eq "1002"
                                       and gid eq "-1" then permit as root
  netbsd-fswrite: filename eq "/var/log/httpd/error_log" then permit as root
  netbsd-fswrite: filename eq "/var/log/httpd/access_log" then permit as root
  netbsd-bind: sockaddr eq "inet-[0.0.0.0]:80" then permit as root
  netbsd-bind: sockaddr eq "inet-[192.168.0.5]:80" then permit as root
  netbsd-bind: sockaddr eq "inet-[127.0.0.1]:80" then permit as root
  netbsd-fsread: filename match "/var/run/httpd.pid" then permit as root
  netbsd-fswrite: filename eq "/var/run/httpd.pid" then permit as root
  netbsd-fswrite: filename match "/var/run/httpd.lock.*" then permit as root
```

Abbildung 7: Angepasste Apache-Richtline

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  puts("Führe Systrace-überwachte Shell aus");
  execlp("/bin/systrace","systrace","-a","-i","/bin/ksh",NULL);
  return 0;
}
```

Abbildung 8: Kapselung des Shell-Aufrufes

Um die Richtlinie automatisch zu erstellen, meldet man sich mit `login systraced` an und starten eine Shell mit `systrace -A /bin/ksh`. Nach Beendigung der Shell liegt das Grundgerüst der Richtlinie in `systraced/.systrace/bin_ksh` vor. Nach Aktivierung der Kapsel aus Abbildung 8 als Shell für den Benutzer erfolgt der `login(1)` unter der UID systraced≪. Da nun die Systrace-Überwachung aktiv ist, landen Systemaufrufe und Fehler im `syslog(3)`. Es ist in der Anfangsphase recht praktisch, in einem weiteren X-Terminal `tail -f /var/log/messages` laufen zu lassen, um so in Echtzeit die fehlgeschlagenen Systemaufrufe analysieren zu können. Die Richtlinien unter `systraced/.systrace/` sind mit `chown(8)` Root und Wheel zuzuordnen und mit `chmod(1)` auf 644 oder gar 444 zu setzen. Selbiges gilt für das `.systrace`-Verzeichnis, allerdings mit den Rechten 755.

In der Beispielrichtlinie aus Abbildung 9 befinden sich vier Blöcke von Ausdrücken. Der erste regelt den Zugriff auf verschiedene Geräte und Konfigurationsdateien, außerdem den Zugriff auf `/tmp/`, `/var/tmp/` und das Heimatverzeichnis des Benutzers.

Im zweiten Block stehen verschiedene Richtlinien, die den Zugriff auf Sockets (Port 22) mittels regulärer Ausdrücke bestimmen.

Der dritte Block widmet sich den `exec(3)`- und `execve(2)`-Aufrufen. Zugriff auf `/usr/bin/ftp` und `/usr/bin/telnet` sowie auf Programme, die in `/home/systraced/` liegen, sind verboten, alle anderen erlaubt. Der Benutzer kann somit keine Programme in seinem Heimatverzeichnis ablegen und von dort aus starten. Er kann allerdings gemäß dieser Richtlinie dies in `/tmp/` oder `/var/tmp/` erledigen.

Im letzten Block befinden sich alle Systemaufrufe ohne Argumente, die Systrace automatisch während des vorhergehenden Initialisierungslaufes geschrieben hat. Es handelt sich dabei um alle Aufrufe, die quasi „im Hintergrund“ laufen und von den aufgerufenen Programmen benötigt werden.

Fazit

Systrace eignet sich nicht nur zur Absicherung eines Systems, sondern auch zur einfachen Überwachung und Analyse von Programmen.

Indem man automatisch eine Richtlinie erstellen lässt, erfährt man, welche Systemaufrufe das Programm durchführt. Diese Richtlinie kann man zu Testzwecken manipulieren und so überprüfen wie sich ein Programm verhält, wenn es beispielsweise nicht mehr auf bestimmte Dateien zugreifen darf. Diese Vorgehensweise ist besonders nützlich bei nur binär verfügbaren Programmen wie Opera oder Acrobat Reader.

Systrace ist ein umfangreiches Programm, das bei korrekter Konfiguration die Sicherheit eines Systemes dramatisch erhöhen kann. Mit Systrace lassen sich Dienste als nichtprivilegierter Benutzer ausführen. Nur bestimmte Systemaufrufe müssen Root-Rechte erhalten. Damit ist das potentielle Schadensrisiko, das von einem SETUID-Programm ausgeht, dramatisch reduziert. Weiterhin

```
Policy: /bin/ksh, Emulation: netbsd
## Zugriffe auf Konfigurationsdateien erlauben
netbsd-fsread: filename eq "/etc/man.conf" then permit
netbsd-fsread: filename eq "/etc/passwd" then permit
netbsd-fsread: filename match "/etc*" then deny
netbsd-fsread: filename match "/home/systraced/*" then permit
netbsd-fsread: filename match "/home*" then deny
netbsd-fsread: filename match "/tmp/*" then permit
netbsd-fsread: filename match "/var/tmp/*" then permit
netbsd-fswrite: filename match "/dev/tty" then permit
netbsd-fswrite: filename match "/tmp/*" then permit
netbsd-fswrite: filename match "/var/tmp/*" then permit
netbsd-fswrite: filename eq "/dev/crypto" then permit

## SSH auf bestimmte Adressen erlauben
netbsd-socket: sockdom eq "AF_INET" and socktype match "*" then permit
netbsd-connect: sockaddr eq "inet-[127.0.0.1]:22" then permit log
netbsd-connect: sockaddr re "inet-.192.168.\[0-8\].\[4-6\].:22" then permit log
netbsd-connect: sockaddr match "inet-192.168.0.\[1-9\].:22" then permit log

netbsd-setuid: uid eq "1007" and uname eq "systraced" then permit
netbsd-seteuid: uid eq "1007" and uname eq "systraced" then permit
netbsd-recvfrom: permit
netbsd-setsockopt: permit

## Ausführbare Dateien in $HOME und ftp + telnet verbieten, sonst erlauben
netbsd-exec: filename sub "/home/systraced" then deny log
netbsd-execve: filename sub "/home/systraced" then deny log
netbsd-execve: filename eq "/usr/bin/ftp" then permit log
netbsd-execve: filename eq "/usr/bin/telnet" then permit log
netbsd-exec: true then permit log
netbsd-execve: true then permit log

netbsd-mmap: permit
netbsd-fsread: permit
netbsd-__fstat13: permit
netbsd-close: permit
[...]
```

Abbildung 9: Kapselung des Shell-Aufrufes

ermöglicht Systrace eine feinere Granulierung von Programmaufrufen oder Sockets man kann beispielsweise Verbindungen des SSH-Clients nur auf bestimmte IP-Adressen erlauben und auf alle anderen verbieten. Der Autors hat beispielsweise damit Gateways eingerichtet, auf denen sich Mitarbeiter von außen anmelden können. Auf den Gateways wird der SSH-*Client* überwacht, so das sich diese Benutzer nur mit bestimmten, eigens freigegebenen, IP-Adressen der Arbeitsgruppenserver verbinden können.

Die Konfiguration einer Systrace-Richtlinie setzt allerdings gute Kenntnisse des jeweiligen Betriebssystems – insbesondere der Systemaufrufe – voraus. Vergessen sollte man auch nicht, dass Systrace eine technische Maßnahme ist, die ohne umfassende Sicherheitsvorgaben der Firma / des Betriebs allein zu kurz greift. *(dw)*

Stefan Schumacher beschäftigt sich in seiner Freizeit mit japanischen Kampfkünsten sowie mit NetBSD und PostgreSQL. Seine persönlichen Webseiten sind unter www.net-tex.de bzw. www.cryptomancer.de erreichbar. Seine OpenPGP-Schlüssel-ID ist 0xB3FBAE33.

Literaturverzeichnis

1. Niels Provos, *Improving Host Security with System Call Policies*, http://www.citi.umich.edu/articles/reports/citi-tr-02-3.pdf
2. Ken Thompson, *Reflections on Trusting Trust*, Communication of the ACM Vol. 27, No. 8, August 1984, http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf
3. Matthias Peterman, Systrace-Restricted Login-Shell mit NetBSD, http://wiki.bsd-crew.de/index.php/Systrace-Restricted_Login-Shell_mit_NetBSD