Systeme mit Systrace härten

Stefan Schumacher <stefan@net-tex.de>

Systrace ermöglicht die Überwachung und Steuerung von Systemaufrufen. Dazu benutzt es Richtlinien, die für jedes verwendete Programm definiert werden. Anhand dieser Richtlinie werden Systemaufrufe erlaubt oder verboten. Ebenso kann man einzelne Systemaufrufe unter anderen Benutzerrechten ausführen. Somit ist es möglich, SETUID-Programme als unprivilegierter Benutzer zu starten und nur bestimmte Systemaufrufe mit Root-Rechten auszuführen. 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 von zwei Beispielen auf NetBSD.

Systemaufrufe mit Systrace steuern

Wie inzwischen allgemein anerkannt ist, setzt eine erfolgreiche Verteidigung eines Rechner-Systems mehrere Verteidigungslinien voraus. Diese Linien müssen sich überlappen, ohne jedoch voneinander abzuhängen.

Dies können beispielsweise Paketfilter, Application-Level-Gateways oder biometrische Zugangskontrollen sein. Allerdings bietet jede Verteidigungslinie wiederum neue Angriffspunkte. So könnte beispielsweise ein Einbruchserkennungssystem mittels Speicherüberlauf übernommen und mit Root-Rechten mißbraucht werden. Daher ist es notwendig, die Schadmöglichkeiten von Programmen einzugrenzen.

Nahezu jedes heute eingesetzte Programm ist zu komplex und umfangreich, um sorgfältig auf Fehlerquellen und Sicherheitslücken überprüft zu werden. Selbst wenn es als quelloffenes Programm vorliegt, überprüft in der Regel niemand den Quellcode auf absichtliche oder fahrlässige Sicherheitsprobleme. Quelloffenheit ist zwar ein sehr gutes Kriterium für sicherheitsrelevante Programme, schützt aber definitiv nicht vor Lücken oder Hintertüren, wie (Thompson, 1984) beweist. Teilweise ist man aber auch auf den Einsatz geschlossener Software angewiesen und hat dann keinerlei Möglichkeit mehr, diese zu prüfen und muß ihr zwangsläufig trauen. Angriffe gegen Systeme konzentrieren sich in der Regel auf Systemaufrufe (auch System Calls oder Syscalls genannt), die unter anderem auch dazu verwendet werden können, um im Kernel privilegierte Operationen durchzuführen. Um derartigen Vorgehensweisen zu begegnen, wird die Angriffsfläche reduziert. Dazu wird ein Programm eingeführt, das die Ausführung von Systemaufrufen begrenzt und die Zugriffsrechte darauf feiner als bisher granuliert.

Hierzu benötigt man eine Richtlinie, die die Zugriffe reglementiert. Die Richtlinie muß alle möglichen Fälle abdecken und auch Kenntnisse von allen möglicherweise auftretenden Pfaden 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 auch Überwachungsprotokolle der überwachten Programme erzeugen, um so Anomalien erkennen und analysieren zu können.

Ein derartiges System läßt sich auf verschiedene Arten implementieren. Einerseits kann es komplett im Kernel arbeiten, andererseits auch kom-

plett im User-Space. Im Kernel ist die Ausführung recht schnell, aber das System selbst ist äußerst komplex und nur sehr schwer auf andere Betriebssysteme zu portieren. Im User-Space hingegen ist das Programm portabel, aber sehr langsam und unsicher, da es zu Race-Conditions zwischen dem Zeitpunkt der Analyse und der Ausführung eines Systemaufrufes kommen kann.

Systrace, von seinem Entwickler Niels Provos in (Provos, 2006) und (Eriksen und Provos, 2003) 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 sehr schnelle Behandlung von kontextinsensitiven Systemaufrufen, die bspw. 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 kontextsensitive Entscheidungen und trifft sie anhand der definierten Richtlinie. Während der Entscheidungsfindung blockiert der Kernel den fraglichen Prozeß. Weiterhin kann der User-Space-Dæmon über eine Schnittstelle Informationen wie Zustandsübergänge, PID-Änderungen oder Forks vom Kernel-Teil anfordern.

Um Funktionen des Kernels zu nutzen, können Anwenderprogramme Systemaufrufe verwenden. Mit diesen kann sich ein Programm bspw. an einen Port binden oder eine Logdatei öffnen, da solche Operationen System-Rechte erfordern. Damit ein Programm auf derartige Systemaufrufe zugreifen darf, muß es mit Root-Rechten gestartet werden – und eröffnet damit eine riesige Sicherheitslücke. Systrace von Niels Provos umgeht dieses Problem, indem die Rechtezuteilung nicht mehr auf Programmebene vorgenommen, sondern auf Ebene der Systemaufrufe heruntergebrochen wird.

Ein Programm kann also von einem normalen Benutzer gestartet werden und bekommt gemäß einer vorher erstellten Richtlinie von Systrace entsprechende Zugriffsrechte auf Systemaufrufe zugeteilt.

Die Grammatik der Richtlinie

Die Richtlinie für ein zu überwachendes Programm wird in einer recht einfachen Grammatik definiert, in der Ausdrücke aneinandergereiht werden. Ein Ausdruck besteht dabei aus einem booleschen Ausdruck und der auszuführenden Aktion. Für die Aktion gibt es folgende Kommandos: *ask* (frage), *deny* (verbiete) oder *permit* (erlaube) in Verbindung mit optionalen Argumenten. Ergibt der boolesche Ausdruck `wahr`, wird die definierte Aktion ausgeführt. Ist die Aktion als *ask* definiert, wird der Benutzer befragt, 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 den Regeln in fnmatch(3) entspricht.
- `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.

Ein paar Ausdrücke zur Veranschaulichung der Möglichkeiten:

- netbsd-execve: `permit` – Erlaubt alle execve(2)-Aufrufe.
- netbsd-execve: `true then permit log` – Erlaubt alle execve(2)-Aufrufe und protokolliert sie mit `syslog`. Das »`true then`« ist nötig, damit »log« eingesetzt werden kann.
- 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 mit `syslog`.
- netbsd-seteuid: `uid eq ''1007'' or uname eq ''systraced'' then permit` – Erlaubt das Setzen der effektiven UID, wenn die UID des aufrufenden Benutzer 1007 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. Auch dieser Aufruf wird protokolliert.

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. Wird kein passender Ausdruck gefunden, wird die Entscheidung an den Benutzer delegiert. Wird ein Systemaufruf abgelehnt, kann Systrace an das aufrufende Programm einen spezifizierten Fehlercode zurückgeben.

Um die Richtlinie auf Benutzer- bzw. Gruppenebene granulieren zu können, werden Ausdrükke 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 wird der Zugriff auf die Datei `/etc/master.passwd` mit dem Fehlercode EPERM abgelehnt, wenn der aufrufende Benutzer nicht Mitglied der Gruppe »wheel« ist.

Soll ein Systemaufruf unter anderen Benutzerrechten ausgeführt werden, kann an den Ausdruck die Direktive as user:group angehängt werden. Beispiele dazu werden später im Text aufgeführt.

An jeden Ausdruck einer Richtlinie kann die Log-Direktive log angehängt werden. Damit wird der Ausdruck und die durch ihn implizierte Entscheidung vom Betriebssystem geloggt. 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 und zu erlauben. Nicht erfaßte Aufrufe sind als Angriff zu werten und zu verbieten. Somit läßt sich eine Richtlinie erstellen, indem ein spezielles Programm die zu erfassende Anwendung bei einem Probelauf überwacht und die abgesetzten Systemaufrufe mitschneidet. Diese werden dabei in die kanonische Form normalisiert und in Richtlinien-Ausdrücke übersetzt. Existiert in der bisherigen Richtlinie kein Ausdruck, der den aktuellen Systemaufruf behandelt, wird ein neuer Ausdruck angehängt, der den Aufruf erlaubt.

Manuelles Eingreifen in die erzeugte Richtlinie ist in der Regel nicht erforderlich, es sei denn, die überwachte Anwendung arbeitet mit Zufallsnamen für Dateien. Dann muß der entsprechende Ausdruck dahingehend abgeändert werden. Bei dieser automatisierten Richtlinienerstellung wird davon ausgegangen, daß das zu überwachende Programm per se sicher ist. Kann dies nicht gewährleistet werden, ist eine so erstellte Richtlinie nicht als sicher zu betrachten. Außerdem ist die Automatik eben-

falls nicht anwendbar, wenn es nicht möglich ist, alle auftretenden Code-Pfade durchzuexerzieren. 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.

Nachdem eine Richtlinie fertiggestellt wurde, kann sie von Systrace gegenüber dem gewünschten Programm durchgesetzt werden.

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 wird ebenfalls ein Programm überwacht und eine Richtlinie erzeugt, allerdings wird der Benutzer bei jedem Systemaufruf um Zustimmung gebeten. Dies ist sinnvoll, wenn dem zu überwachenden Programm nicht unbedingt von vornherein vertraut werden 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. Nicht erlaubte Systemaufrufe werden abgelehnt und protokolliert.

Systrace verfügt über eine Reihe von Optionen:

- `-A` Initialisierungsmodus, erzeugt eine Richtlinie, in der alle Systemaufrufe erlaubt sind.
- `-a` Überwachungsmodus, setzt die definierte Richtlinie durch.
- `-c UID:GID` Spezifiziert eine numerische User- und Gruppen-ID. Unter diesen wird das zu überwachende Programm ausgeführt. 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 eine alternative GUI.
- `-i` Vererbt die Richtlinie des Elternprozesses an die Kinder.
- `-p pid` 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 selbst hat auch mit einigen Sicherheitsproblemen zu kämpfen, z.B. Aliasen auf Systemressourcen, Dateinamen von Programmen, die in einem Chroot laufen oder der Verfolgung von Prozess-IDs.

Der Zugriff auf ein und dieselbe Datei ist unter Unix dank symbolischer Links und relativer Pfadnamen auf unendlich viele Arten möglich. Außerdem können Dateien von verschiedenen Diensten, wie z.B. Proxies, NFS oder CFS, bereitgestellt werden. Derartige Dienste sind für Systrace nicht sichtbar, müssen 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 Parame-

```
1 # systrace -A apachectl start
2 /usr/pkg/sbin/apachectl start: httpd started
3 # apachectl stop
4 /usr/pkg/sbin/apachectl stop: httpd stopped
5 # ls /root/.systrace/
6 usr_pkg_sbin_apachectl usr_pkg_sbin_httpd
```

Abbildung 2: Automatisch eine Richtline für Apache erstellen

ter für Systemaufrufe werden von Systrace normalisiert, indem Dateinamen in absolute Form – also ohne Symlinks oder relative Pfadangaben – umgewandelt werden. Diese normalisierten Werte werden dem Betriebssystem wieder übergeben.

Ausgenommen hiervon sind nur einige Systemaufrufe, wie z.B. `readlink`. Weiterhin werden diese Werte auf einem nicht beschreibbaren Puffer zwischengespeichert, so daß kein anderer Prozeß die Werte manipulieren kann. Der Kernel verweigert die Ausführung von Systemaufrufen, die Symlinks als Argumente enthalten. Somit werden nur noch von Systrace normalisierte Aufrufe, die erlaubt sind, ausgeführt. Alle anderen Aufrufe werden abgelehnt.

Werden Systemaufrufe abgelehnt, müssen die überwachten Programme entsprechende Fehlermeldungen bekommen. Da nicht alle Programme eine funktionierende Fehlerbehandlung implementieren, kann in der Richtlinie ein bestimmter Fehlercode spezifiziert werden.

Ebenfalls zu betrachten ist ein Richtlinien-Wechsel und Prozeß- Beendigung. Wenn ein überwachter Prozeß einen neuen Prozeß startet, wird der alte Prozeß vom System aus dem Speicher entfernt. Der neue Prozeß wird stattdessen ausgeführt. Dieser neue Prozeß kann ein vertrauenswürdiges Programm sein, so daß keine weitere Überwachung notwendig ist. Es kann aber auch ein Prozeß sein, der mit einer anderen Richtlinie besser überwacht wird. Systrace überwacht Systemaufrufe auf Erfolg und kann so nach einem geglückten execve-Systemaufruf den neuen Prozeß mit einer anderen Richtlinie überwachen oder die Überwachung beenden.

Die gesamten Richtlinien werden 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-Fileflags (`schg`) in Verbindung mit den Security-Leveln verwenden. Möchte man diese Methode nicht einsetzen, sollten die

```
1 [...]
2 netbsd-pread: permit
3 netbsd-fsread: filename eq „/etc/group“ then permit
4 netbsd-fsread: filename eq „/usr/pkg/etc/httpd/httpd.conf“ then permit
5 netbsd-fsread: filename eq „/usr/pkg“ then permit
6 netbsd-fsread: filename eq „/usr/pkg/etc/httpd/srm.conf“ then permit
7 netbsd-fsread: filename eq „/usr/pkg/etc/httpd/access.conf“ then permit
8 netbsd-gettimeofday: permit
9 netbsd-fsread: filename eq „/etc/etc.network/resolv.conf“ then permit
10 netbsd-fsread: filename eq „/etc/hosts“ then permit
11 netbsd-chmod: filename eq“/var/run/httpd.mm.2872.sem“
12 and mode eq „600“ then permit
13 netbsd-chown: filename eq“/var/run/httpd.mm.2872.sem“
14 and uid eq „1002“ and gid eq „-1“ then permit
15 netbsd-fswrite: filename eq „/var/log/httpd/error_log“ then permit
16 netbsd-dup2: permit
17 netbsd-select: permit
18 netbsd-fsread: filename eq „/usr/pkg/etc/httpd/mime.types“ then permit
19 netbsd-fsread: filename eq „/usr/pkg/etc/httpd/magic“ then permit
20 netbsd-fswrite: filename eq „/var/log/httpd/access_log“ then permit
21 netbsd-chdir: filename eq „/“ then permit
22 netbsd-fork: permit
23 netbsd-exit: permit
24 netbsd-setsid: permit
25 netbsd-fsread: filename eq „/dev/null“ then permit
26 netbsd-fswrite: filename eq „/dev/null“ then permit
27 netbsd-socket: sockdom eq „AF_INET“ and socktype eq „SOCK_STREAM“ then
permit
28 netbsd-setsockopt: permit
29 netbsd-bind: sockaddr eq „inet-[0.0.0.0]:80“ then permit
30 netbsd-listen: permit
31 netbsd-bind: sockaddr eq „inet-[192.168.0.5]:80“ then permit
32 netbsd-bind: sockaddr eq „inet-[127.0.0.1]:80“ then permit
33 netbsd-fsread: filename eq „/var/run/httpd.pid“ then permit
34 netbsd-umask: permit
35 netbsd-fswrite: filename eq „/var/run/httpd.pid“ then permit
36 netbsd-write: permit
37 netbsd-fswrite: filename eq“/var/run/httpd.lock.1827“ then permit
38 [...]
```

Abbildung 3: Automatisch erzeugte Richtlinie für Apache (Auszug)

Richtlinien zumindest regelmäßig mit einem Integritätsprüfer wie mtree(8), Aide oder Tripwire überprüft werden.

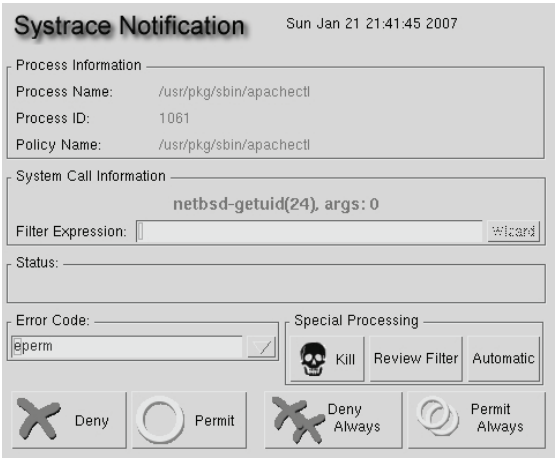
```
1 [...]
2 netbsd-chmod: filename match“/var/run/httpd.mm.*.sem“
3 and mode eq „600“ then permit
4 netbsd-chown: filename match“/var/run/httpd.mm.*.sem“
5 and uid eq „1002“ and gid eq „-1“ then permit
6 netbsd-fswrite: filename match“/var/log/httpd/error_log“ then permit
7 [...]
8 netbsd-fswrite: filename match“/var/run/httpd.lock.*“ then permit
9 [...]
```

Abbildung 4: Automatisch erzeugte Richtlinie korrigiert für Apache (Auszug)

Apache überwachen

Mit den Befehlen aus Abbildung 2 wird Apache gestartet. Dabei wird er von Systrace überwacht, so daß eine Richtlinie erstellt wird. Diese Richtlinie wird geschrieben, nachdem Apache wieder beendet wurde. Ein Teil der Richtlinie wird in Abbildung 3 gezeigt. In den Zeilen 11, 13 und 37 wird auf eine Datei im Verzeichnis `/var/run` zugegriffen. Wie man leicht erkennt, enthält der Dateiname die Prozeß-ID bzw. Semaphoren-Nummern.

Diese müssen durch den Joker »*« ersetzt werden. Außerdem muß der Operator »`eq`« durch »`match`« ersetzt werden, wie in Abbildung 4 gezeigt wird. In den Zeilen 31 und 32 bindet sich Apache an die angegebenen IP-Adressen und 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 Lese-Zugriff für `/home/www` beinhaltet. Die Verstöße werden in `/var/log/messages` protokolliert. Um dieses Problem möglichst komfortabel zu beheben, wird Apache wieder im Initialisierungsmodus von Systrace gestartet: systrace apachectl start. Wird nun in einem Browser die Adresse `http://127.0.0.1/` aufgerufen, meldet sich xsystrace (Ein Beispiel findet sich in Abbildung 1) zu Wort und verlangt vom Benutzer die Bestätigung oder Verweigerung der benötigten Systemaufrufe.



Da diese Prozedur für jede einzelne von Apache geladene Datei durchlaufen wird, brechen wir nach dem Laden der Index-Seite ab und beenden den Apache mit apachectl stop. Systrace hat die Richtlinie bereits um die Lesezugriffe auf `/home/www` erweitert. Allerdings für jede Datei einzeln, so daß hier wieder Reguläre Ausdrücke mit »match« und »*« eingesetzt werden. Abbildung 5 zeigt die automatisch generierte Richtlinie, die jede aufgerufene Datei einzeln behandelt. In Abbildung 6 wurde die Richtlinie mit regulären Ausdrücken verfeinert, so daß Zugriffe auf `/home/www/public/*` gestattet und auf `/home/www/institutsintern/*` verboten werden. Weiterhin wurden die Zugriffe auf CGI-Dateien erlaubt bzw. verboten. Zugriffe auf die verbotenen Dateien werden mit ENOENT abgelehnt, so daß für Apache die Dateien nicht existieren. Bis jetzt wurde die Richtlinie soweit konfiguriert, daß Apache unter Überwachung normal funktioniert.

Trotzdem muß Apache noch als `root` gestartet werden. Systrace kann mit der Option `-c` eine beliebige numerische Benutzer- und Gruppen-ID übernehmen, als die der zu überwachende Prozeß ausgeführt werden soll. Dazu muß allerdings noch die

Richtlinie angepaßt werden, da ein nicht privilegierter Benutzer keine privilegierten Operationen ausführen darf. Systrace kann jede Aktion in der Richtlinie als ein anderer Benutzer ausführen. Somit kann eine Anwendung als normaler Benutzer ausgeführt werden, und nur die benötigten Systemaufrufe werden als root ausgeführt. Dann müssen Systrace und das zu überwachende Programm aber als root gestartet werden. Startet man Apache nun mit systrace -c 1002:1001 apachectl start als Benutzer und Gruppe »www« mit der bisherigen Richtlinie, wird sich Systrace erneut zu Wort melden. Es werden alle Systemaufrufe angezeigt, die als Benutzer »www« nicht ausgeführt werden können. Am einfachsten editiert man daher vorher die Richtlinie mit vi(1) und sucht zuerst nach Aktionen, in denen Dateien mit »netbsd-fswrite« geschrieben werden. Das umfaßt in diesem Beispiel die Log-, PID- und Semaphoren-Dateien.

```
1 [...]
2 netbsd-fsread: filename eq „/home/www“ then permit
3 netbsd-fsread: filename eq „/home/www/.htaccess“ then permit
4 netbsd-fsread: filename eq „/home/www/index.html“ then permit
5 netbsd-fsread: filename eq „/home/www/index.pl“ then permit
6 netbsd-fsread: filename eq „/home/www/index.mhtml“ then permit
7 [...]
```

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

Die nächsten offensichtlichen Kandidaten sind alle »netbsd-bind«-Aktionen, in denen sich Apache an die Netzwerkgeräte bindet. Die angepaßten Aktionen der Richtlinie für nicht-privilegierte Läufe finden Sie in Abbildung 7. Mit dieser Richtlinie kann Apache via Systrace als Benutzer »www« gestartet werden.

Systrace-überwachte Shell

Matthias Petermann beschreibt in (Peterman, 2005) die Einrichtung einer Shell, die von Systrace komplett überwacht wird.

Um eine Shell von Systrace überwachen zu lassen, müßte die Shell über Systrace in der Art systrace -a -i ksh gestartet werden. Da solch ein Befehl nicht in der /etc/master.passwd eingetragen werden kann, muß er in einem kleinen C-Programm (Abbildung 8) gekapselt werden. Diese Kapsel wird nach Erstellung der Richtlinie in /etc/shells und /etc/master.passwd für die jeweils zu überwachenden Benutzer als Shell eingetragen. In diesem Beispiel soll der Benutzer systraced eine eingeschränkte Korn-Shell erhalten.

Um die Richtlinie automatisch zu erstellen, loggen wir uns mit login systraced ein und starten eine Shell mit systrace -A /bin/ksh. Nach Beendigung der Shell haben wir wieder das Grundgerüst unserer Richtlinie in systraced/.systrace/bin_ksh vorliegen. Nun wird die Shell-Kapsel aus Abbildung 8 als Shell für den Benutzer aktiviert. Zudem wird sich wieder mit login(1) als systraced eingeloggt. Da nun die Systrace-Überwachung aktiv ist, werden Systemaufrufe überwacht und Fehler via syslog(3) protokolliert. Es ist in der Anfangsphase recht praktisch, in einem weiteren XTerminal tail -f /var/log/messages laufen zu lassen, um sofort 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 Beispiel-Richtlinie aus Abbildung 9 befinden sich vier Blöcke von Ausdrücken. Im ersten Block wird der Zugriff auf verschiedene Geräte und Konfigurationsdateien geregelt. Außerdem wird der Zugriff auf /tmp/, /var/tmp/ und das Heimatverzeichnis des Benutzers geregelt.

```
1 [...]
2 netbsd-fsread: filename match „/home/www/public/*“ then permit
3 netbsd-fsread: filename match „/home/www/institutsintern/*“
4 then deny [enoent]
5 netbsd-fsread: filename eq „/home/www/cgi-bin/cvsweb.cgi“ then permit
6 netbsd-fsread: filename eq „/home/www/cgi-bin/postgresql.cgi“
7 then deny [enoent]
8 [...]
```

Abbildung 6: Angepasste Apache-Richtline