Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher und Jan W. Meine
Meine Verlag Magdeburg

Systemcalls mit Systrace steuern

Stefan Schumacher

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 unpriviliegierter Benutzer auszuführen und nur bestimmte Systemaufrufe mit Root-Rechten auszuführen. Systrace erlaubt daher die Implementierung einer feingranulierten Sicherheitsrichtlinie, die sogar Argumente von Systemaufrufen überprüfen kann.

Schlüsselwörter: Systrace, Syscalls, Kernel, Security, SETUID, Root-Rechte

**Zuerst erschienen als: S. Schumacher, Systeme mit Systrace härten in Proceedings des GUUG Frühjahrsfachgespräches 2007, Lehmanns Media Berlin*

Zitationsvorschlag: Schumacher, Stefan (2011). Systemcalls mit Systrace steuern. Magdeburger Journal zur Sicherheitsforschung, Band 2, 2011, S. 125–138.
http://www.wissens-werk.de/index.php/mjs/article/viewFile/109/94

Systemaufrufe mit Systrace steuern

Die erfolgreiche Verteidigung eines Rechner-Systems setzt 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 überwachtenProgramme 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 komplett 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, vom 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 kontextsensiti-

ve 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 wird, 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. Die Aktion kann aus folgenden Aktionen bestehen: *ask* (frage), *deny* (verbiete) oder *permit* (erlaube) in Verbindung mit optionalen Argumenten. Ergibt der boolesche Ausdruck *wahr*, wird die definierte Aktion ausgeführt. Ist als Aktion *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 gemäß den Regeln in fnmatch(3) getroffen wird.
- *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 an 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 an 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 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. 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 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 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 im Kapitel 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, 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 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 au-

tomatisierten Richtlinienerstellung wird davon ausgegangen, daß das zu überwachende Programm per sè sicher ist. Kann dies nicht gewährleistet werden, 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. 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 Richtline 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` 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 kön-

nen 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 Parameter 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 `readlink`. Weiterhin werden diese Werte auf einem nurlesbaren Puffer zwischengespeichert, sodaß 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 normalisierteAufrufe, 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, sodaß 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.

Mithilfe des `log`-Befehls kann in einer Richtlinie jeder ausgeführte Systemaufruf protokolliert werden. Somit ist es möglich, mit einer Systrace-überwachten Shell alle `execve`-Aufrufe eines Benutzers zu überwachen. Hierbei muss man aber unbedingt datenschutzrechtliche und andere rechtliche Regelungen beachten.

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

Apache überwachen

Mit den Befehlen aus Abbildung 2 wird Apache gestartet. Dabei wird er von Systrace überwacht, sodaß eine Richtli-

nie 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 Apache mit `apachectl stop`. Systrace hat die Richtlinie bereits um die Lesezugriffe auf /home/www erweitert. Allerdings für jede Datei einzeln, sodaß 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, sodaß 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 das für Apache die Dateien nicht existieren. Bis jetzt wurde die Richtlinie soweit konfiguriert, das 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. 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 . Mit dieser Richtline 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 und 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 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 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.

Im zweiten Block sind verschiedene Ausdrücke, die die Erstellung von Sockets regeln. Dabei werden mit den Regulären Ausdrücken die Zugriffe auf verschiedene IP-Adressen bzw. Adress-Bereiche unter Port 22 erlaubt. Alle derartige Ausdrücke werden protokolliert.

Im dritten Block werden die exec(3)- und execve(2)-Aufrufe behandelt. Aufrufe von /usr/bin/ftp und /usr/bin/telnet sowie von Programmen, die in /home/systraced/, liegen werden verboten, alle anderen erlaubt. Der Benutzer kann somit auch keine Programme in seinem Benutzerverzeichnis ablegen und von dort aus starten. Er kann allerdings in dieser Richtlinie noch Programme in /tmp/ oder /var/tmp/ ausführen.

Im letzten Block finden sich alle Systemaufrufe, die ohne Argumente aufgeführt werden und von Systrace während des ersten Initialisierungslaufes geschrieben wurden.

Weiterer Nutzen

Systrace eignet sich nicht nur zur Absicherung eines Systemes, sondern auch zur einfachen Überwachung und Analyse von Programmen. Indem man automatisch eine Richtlinie erstellen läßt, erfährt man welche Systemaufrufe vom Programm durchgeführt werden. 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 sehr nützlich bei großen Programmen, die nur als Binärversion verfügbar sind, wie Opera oder Acrobat Reader. Ebenso kann man damit selbst entwickelte Programme auf Fehlerbehandlungen hin überprüfen.

Fazit

Systrace ist ein umfangreiches Programm, das bei korrekter Konfiguration die Sicherheit eines Systemes dramatisch erhöhen kann. Mit Systrace lassen sich Dienste als nicht-privilegierter Benutzer ausführen. Nur bestimmte Systemaufrufe müssen Root-Rechte erhalten. Damit ist das potentielle Schadensrisiko, das von einem SETUID-Programm ausgeht, dramatisch reduziert. Weiterhin ermöglicht Systrace eine feinere Granulierung von Programmaufrufen oder Sockets man kann beispielsweise SSH nur auf bestimmte IP-Adressen erlauben und auf alle anderen verbieten.

Die Konfiguration einer Systrace-Richtlinie setzt allerdings gute Kenntnisse des Betriebssystemes – insbesondere der Systemaufrufe – und der Anwendung voraus. Außerdem ist Systrace eine technische Maßnahme, die ohne umfassende Sicherheitsrichtlinie allein zu kurz greift.

Weitere Literatur finden Sie in Ioannidis, Bellovin und Smith (2006) und Goldberg, Wagner, Thomas u. a. (1996).

Über den Autor

Stefan Schumacher ist Direktor des Magdeburger Instituts für Sicherheitsforschung und Mit-Herausgeber des Magdeburger Journals zur Sicherheitsforschung. Er beschäftigt sich seit ca. 15 Jahren mit freien Unix-Systemen, insbesondere NetBSD.

Literaturverzeichnis

Eriksen, M. A. & Provos, N. (2003). Enges Korsett: Systrace setzt Regeln für erlaubte Systemaufrufe durch. *Linux Magazin, 2003*. Zugriff am unter `http://www.www.de/url`

Goldberg, I., Wagner, D., Thomas, R. (1996). A Secure Environment for Untrusted Helper Applications. *Proceedings of the 6th Usenix Security Symposium*.

Ioannidis, S., Bellovin, S. M. & Smith, J. M. (2006). Sub-Operating Systems: A New Approach to Application Security. In *Proceedings of the SIGOPS European Workshop*. SIGOPS@.

Peterman, M. (2005). Systrace-Restricted Login-Shell mit NetBSD. Zugriff am 21. Januar 2007, unter `http://wiki.bsd-crew.de/index.php/Systrace-Restricted_Login-Shell_mit_NetBSD`

Provos, N. (2006). Improving Host Security with System Call Policies. Zugriff am 4. November 2006, unter `http://www.citi.umich.edu/articles/reports/citi-tr-02-3.pdf`

Thompson, K. (1984). Reflections on Trusting Trust. In *Communication of the ACM* (Seiten 761–763). Association for Computing Machinery, Inc. Zugriff am 26. Dezember 2006, unter `http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf`

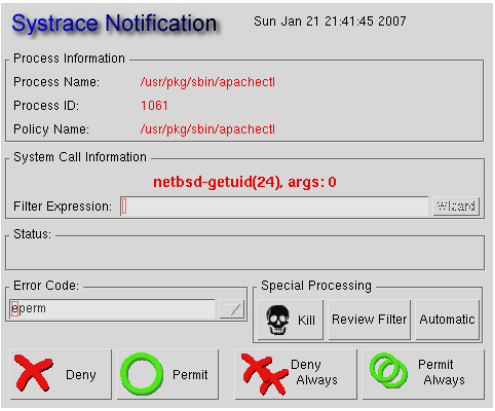

Abbildung 1: Xsystrace bittet den Benutzer um eine Entscheidung

```
# 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 2: 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 3: Automatisch erzeugte Richtlinie für Apache (Auszug)

```
[...]
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 4: Automatisch erzeugte Richtlinie korrigiert für Apache (Auszug)

```
[...]
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/institutsintern/*"
                                    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

```
# 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

```
1  Policy: /bin/ksh, Emulation: netbsd
2  ## Zugriffe auf Konfigurationsdateien erlauben
3  netbsd-fsread: filename eq "/etc/man.conf" then permit
4  netbsd-fsread: filename eq "/etc/passwd" then permit
5  netbsd-fsread: filename match "/etc*" then deny
6  netbsd-fsread: filename match "/home/systraced/*" then permit
7  netbsd-fsread: filename match "/home*" then deny
8  netbsd-fsread: filename match "/tmp/*" then permit
9  netbsd-fsread: filename match "/var/tmp/*" then permit
10 netbsd-fswrite: filename match "/dev/tty" then permit
11 netbsd-fswrite: filename match "/tmp/*" then permit
12 netbsd-fswrite: filename match "/var/tmp/*" then permit
13 netbsd-fswrite: filename eq "/dev/crypto" then permit
14
15 ## SSH auf bestimmte Adressen erlauben
16 netbsd-socket: sockdom eq "AF_INET" and socktype match "*" then permit
17 netbsd-connect: sockaddr eq "inet-[127.0.0.1]:22" then permit log
18 netbsd-connect: sockaddr re "inet-.192\.168\.[0,4,8]\.[4-6].:22"
19         then permit log
20 netbsd-connect: sockaddr re "inet-.192\.168\.0\.5\.:22" then permit log
21 netbsd-setuid: uid eq "1007" and uname eq "systraced" then permit
22 netbsd-seteuid: uid eq "1007" and uname eq "systraced" then permit
23 netbsd-recvfrom: permit
24 netbsd-setsockopt: permit
25
26 ## Ausführbare Dateien in $HOME und ftp/telnet verbieten, sonst erlauben
27 netbsd-exec: filename sub "/home/systraced" then deny log
28 netbsd-execve: filename sub "/home/systraced" then deny log
29 netbsd-execve: filename eq "/usr/bin/ftp" then permit log
30 netbsd-execve: filename eq "/usr/bin/telnet" then permit log
31 netbsd-exec: true then permit log
32 netbsd-execve: true then permit log
33
34 netbsd-mmap: permit
35 netbsd-fsread: permit
36 netbsd-__fstat13: permit
37 netbsd-close: permit
38
39 [...]
```

Abbildung 9: Kapselung des Shell-Aufrufes