

# Systrace: sicurezza alla base...

come rendere sicuro un sistema sfruttando un access manager per le system call

di Matteo Cantoni  
goony@OpenBEER.it

La sicurezza dei sistemi informatici è tradizionalmente un tema molto discusso nell' ampia comunità degli addetti ai lavori come pure negli ambienti accademici, anche se spesso l'effettivo impiego di tecniche volte a migliorare i sistemi stessi sotto il profilo della vulnerabilità risulta essere insufficiente alla prova effettiva dei fatti (vale a dire in presenza di reali attacchi).

Lo studio e l'utilizzo degli strumenti usualmente a disposizione per rendere i nostri sistemi sempre più sicuri (antivirus, firewall, ids, nids, ecc.) può rivelarsi insufficiente, a causa di una ragione di fondo molto importante: la sicurezza ottimale è da considerarsi solo teorica e bisogna partire dal presupposto che qualsiasi software può contenere degli errori, delle vulnerabilità sfruttabili da possibili attaccanti in molteplici maniere e circostanze.

Soluzioni come il Discretionary Access Control dei sistemi Unix-like (ovvero l'usuale sistema di permessi e privilegi utilizzato nella gestione di utenti e risorse) evidentemente non risultano essere più adeguati per certe realtà oggi giorno. Per ovviare questo problema andremo a presentare un interessante meccanismo di sicurezza chiamato Systrace.

Sviluppato da Niels Provos, Systrace è essenzialmente un "system call access manager", le cui basi si fondono sul concetto che per eseguire qualsiasi operazione che coinvolga l'accesso al risorse di un sistema si deve fare affidamento sull'utilizzo di specifiche chiamate di sistema, "system calls" (syscall(2)). L'insieme delle system calls costituisce l'interfaccia che consente ai programmi utente di interagire con il kernel del sistema operativo.

Creando apposite policy possiamo, in caso di compromissione del sistema, limitare i privilegi acquisiti dall'attaccante e i possibili danni che può provocare. Potremmo utilizzare Systrace ad esempio per "hardenizzare" (rendere più sicuro) un server web o in generale una "sandbox" che offre dei servizi su internet. Le funzioni e gli utilizzi che offre Systrace sono molteplici, fra cui:

- confinamento di applicazioni binarie insicure
- generazione interattiva delle policy (sia testualmente che per mezzo di GUI)
- policy enforcement non interattivo
- supporto per emulazioni differenti: GNU/Linux, BSDI, etc.
- riscrittura degli argomenti delle system calls
- monitoraggio remoto e funzionalità di intrusion detection
- elevazione dei privilegi ad esempio per applicazioni con setuid settato

Nel corso dell'articolo cercheremo di capire il suo funzionamento teorico e come riesca ad interporre tra il livello utente e quello kernel, arrivando a studiare qualche policy d'esempio. Al momento della stesura di questo documento le ultime modifiche ai sorgenti di Systrace sono datate 23.06.2003. Systrace può essere utilizzato su diversi sistemi operativi fra cui: Darwin, Linux, NetBSD, OpenBSD. Noi abbiamo scelto di studiare questo potente strumento su un sistema su cui è installato OpenBSD (release 3.3), un sistema operativo Unix-like rivolto alla sicurezza che ha integrato Systrace direttamente all'interno del sistema base già dalla release 3.2. Per un'installazione su altri sistemi operativi vi rimandiamo ai links in fondo all'articolo, tra cui potete trovare un utile "Systrace policy repository".

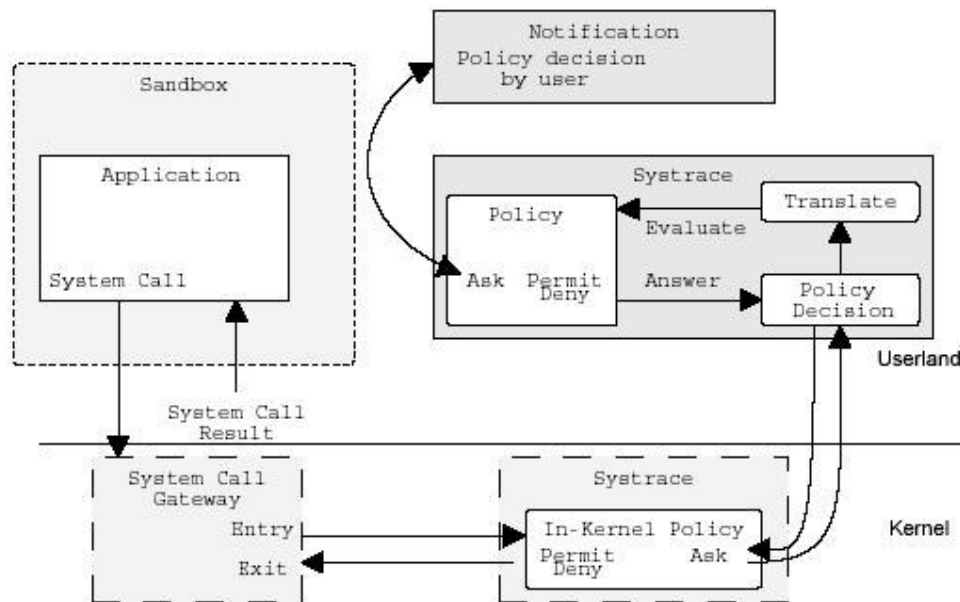
Systrace per motivi di prestazioni, portabilità e complessità è stato sviluppato in due parti, una in user-level e l'altra in kernel-level. Quando un'applicazione monitorata esegue una system call, Systrace a livello kernel controlla un piccolo database contenente le policy, decidendo se permettere o negare l'utilizzo della stessa. Nel caso di un'applicazione complessa, dove le policy presenti non permetterebbero un match con la syscall, Systrace negherebbe automaticamente l'utilizzo di quest'ultima. Vedremo in seguito che in alcuni casi, ad esempio durante la creazione delle policy, l'utente avrà la possibilità di decidere sul destino della system call grazie ad un demone a livello utente (apposita interfaccia testuale o grafica).

Il kernel colloquia direttamente con il demone in user-level per mezzo del device `/dev/systrace`.

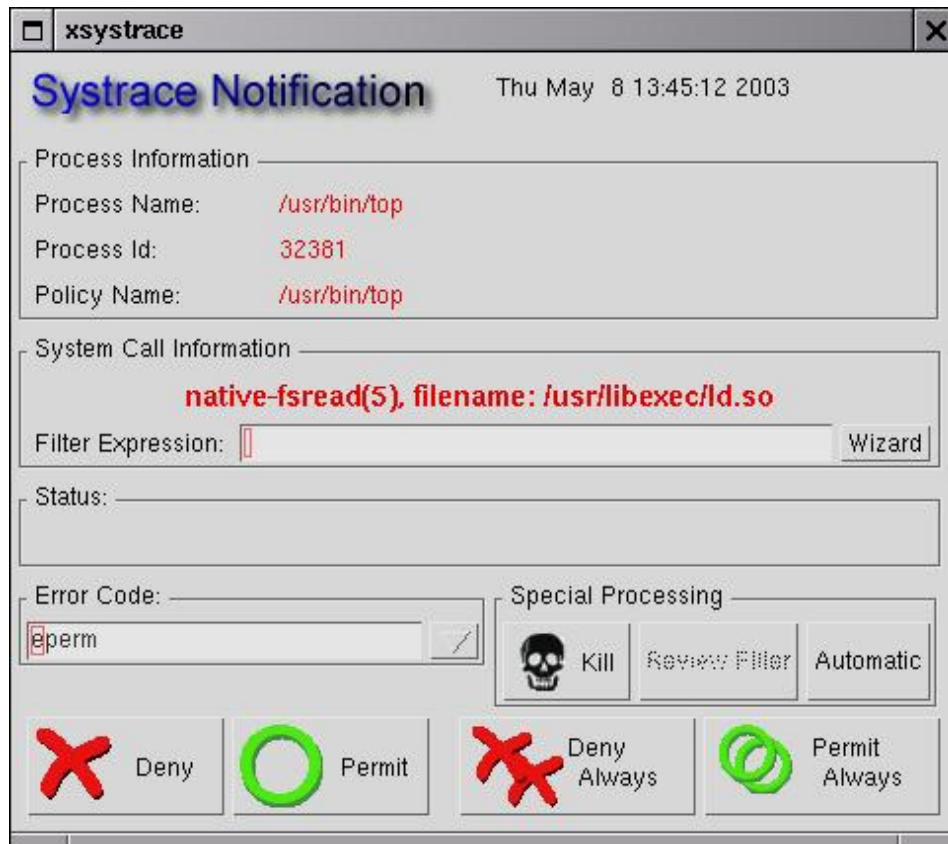
Nel momento in cui il kernel si aspetta una risposta dal demone, sospende il processo che richiede tale system call. Se il processo viene alterato prima che la risposta arrivi al kernel, la system call viene negata ed è restituito un errore. Ogni "transizione" tra il kernel e il demone è sincronizzata per mezzo di un apposito sequence number.

Systrace permette anche di monitorare gli argomenti passati ad una system call per mezzo di una funzione di traduzione e normalizzazione. In questo modo il kernel ha la capacità di riscrivere quest'ultimi prima di permettere l'esecuzione di un processo.

Systrace vieta di default tutte le azioni che non sono esplicitamente permesse. Nel momento in cui una system call viene negata abbiamo la possibilità di restituire a nostro piacimento un codice d'errore (`errno(2)`), gestibile per mezzo del demone `syslogd`.



La generazione delle policy risulta essere l'operazione cruciale e a volte più ostica, anche se la sintassi è, con un po' di pratica, decisamente user-friendly. Le policy possono essere create in tre modalità: manualmente, in modo interattivo testuale e in modo interattivo per mezzo d'una GUI.



In un sistema dove è installato Systerce avrete a disposizione una directory "/etc/systrace" per raccogliere le policy globali utilizzate da ogni utente. Inoltre ogni singolo utente avrà a disposizione nella propria home directory uno spazio "\$HOME/.systrace" per le proprie policy personali. Ovvio che in queste condizioni la sicurezza del filesystem diventa indispensabile; in caso contrario un attaccante potrebbe modificare le policy e compromettere ogni applicazione monitorata.

Ogni policy viene chiamata con il suo path assoluto. Ad esempio una policy per il demone named situato al percorso "/usr/sbin/named" sarà identificata con "usr\_sbin\_named" (notare l'utilizzo del del carattere "\_" invece che "/").

La sintassi:

```
filter = expression "then" action errorcode logcode
  expression = symbol | "not" expression | "(" expression ")" |
  expression "and" expression | expression "or" expression
symbol = string typeoff "match" cmdstring |
  string typeoff "eq" cmdstring | string typeoff "neq" cmdstring |
  string typeoff "sub" cmdstring | string typeoff "nsub" cmdstring |
  string typeoff "inpath" cmdstring | string typeoff "re" cmdstring |
  "true"
typeoff = /* empty */ | "[" number "]"
action = "permit" | "deny"
errorcode = /* empty */ | "[" string "]"
logcode = /* empty */ | "log"
```

dove

"cmdstring" è una qualsiasi stringa arbitraria chiusa tra doppi apici;

"errorcode" è il codice d'errore personalizzabile restituito a syslog;

Gli operatori che si possono utilizzare per il matching delle policy sono:

- match considerata vera se il nome del file coincide con fn-match(3)
- eq considerata vera se l'argomento della system call coincide esattamente con la cmdstring
- neq negazione logica di eq
- sub effettua una verifica del substring nell'argomento della system call
- nsub negazione logica di sub
- inpath considerata vera se l'argomento della system call è un subpath della cmdstring
- re considerata vera se l'argomento della system call verificano la specifica espressione regolare

```
Policy: Abin/ls, Emulation: native
  native-munmap: permit
[...]
  native-stat: permit
  native-fsread: filename match "/usr/" then permit
  native-fsread: filename eq "/tmp" then permit
  native-fsread: filename eq "/etc" then deny[enotdir]
  native-fchdir: permit
  native-fstat: permit
  native-fcntl: permit
[...]
  native-close: permit
  native-write: permit
  native-exit: permit
```

Nelle policy possono essere utilizzate variabili come \$HOME, \$USER e \$PWD, e come di consuetudine le righe di commento iniziano con il carattere "#". Naturalmente la generazione manuale di policy può rivelarsi un'operazione abbastanza ardua e imprecisa. Systrace offre allora due interfacce per semplificare il lavoro. Se volessimo ad esempio creare una policy per il demone named per mezzo di una comoda interfaccia grafica useremmo:

```
systrace /usr/sbin/named
```

Da questo momento ogni system call sarebbe monitorata e l'utente avrebbe la possibilità di decidere il dafarsi. Non ci dilunghiamo troppo nella spiegazione della GUI che risulta essere commentata a dovere. In alternativa è possibile utilizzare un'interfaccia testuale interattiva utilizzando l'opzione "-t" (text mode). Se invece preferissimo far generare delle policy direttamente da Systrace possiamo utilizzare l'opzione "-A" (automatically). In questo modo tutte le system call necessarie al regolare funzionamento del processo monitorato sarebbero permesse e all'amministratore di sistema resterebbe soltanto da "ritoccare manualmente" le policy generate. Una volta generata la policy (posizionata automaticamente in \$HOME/.systrace) potete fermare il processo in esame e riavviarlo testandolo con l'opzione "-a" (automatic) di Systrace. Nel caso precedente utilizzeremmo:

```
systrace -a /usr/sbin/named
```

La generazione di policy automatica anche se molto potente può rivelarsi a volte troppo permissiva e non performante al caso specifico. Utile anche l'opzione "-p" (pid) che permette di monitorare applicazioni già in funzione sul sistema. L'ottimizzazione delle policy spesso può non essere molto intuitiva, e diventa indispensabile la consultazione e lo studio dei log (/var/log/messages).

I tasks che vi possiamo consigliare per il monitoraggio di un sistema generico potrebbero essere:

- installazione del sistema operativo
- ottimizzazione (hardening) del sistema
- configurazione dei servizi di rete
- generazione automatica delle policy tramite Systrace
- ottimizzazione manuale delle policy
- test del servizio di rete in una lan privata
- messa in produzione del servizio di rete

[specchietto o riquadro a lato]

Le altre opzioni principali di Systrace:

- a           abilita l'utilizzo di una policy
- A           generazione automatica di policy
- u           disabilita l'aliasing per le system call
- i           eredita le policy per i processi figli
- t           interfaccia testuale per la generazione interattiva delle policy
- U           ignora le policy personali dell'utente e considera quelli globali
- d policydir   specifica la directory alternativa che contiene le policy
- g gui       permette di specificare una gui alternativa
- c uid:gid    specifica l'uid e il gid con cui sarà eseguita l'applicazione monitorata
- f file       specifica i files delle policy
- p pid       specifica il pid del processo da monitorare

Systrace introduce anche un nuovo meccanismo per la protezioni di quei binari che normalmente vengono avviati con permessi privilegiati. Con il "privilege elevation" specifiche policy possono permettere l'esecuzione di solo alcuni processi avviati da un binario con privilegi elevati, eliminando la necessità di utilizzare binari con setuid e setgid settato. Invece che avviare l'intera applicazione con speciali privilegi, tramite policy assegnamo specifici permessi solo alla singola system call che lo necessita. Sarà poi il kernel che si occuperà di aumentare e decrementare i privilegi prima e dopo l'utilizzo della system call. Grazie a questa modalità, nel momento in cui un attaccante si trovasse a sfruttare una vulnerabilità in un applicazione, si ritroverebbe limitato e non con i privilegi garantiti dal setuid.

La possibilità di effettuare danni risulterebbe in questo modo estremamente ridotta.

```
native-socket: sockdom eq "AF_INET" and socktype eq "SOCK_RAW" then permit as root
native-bind: sockaddr eq "inet-[0.0.0.0]:22" then permit as root
native-fsread: filename eq "/dev/kmem" then permit as :kmem
```

(notare la stringa "then permit as root")

Quindi servizi di rete monitorati da Systrace con questo meccanismo raggiungerebbero un alto grado di sicurezza. Se volete cimentarvi in qualche esperimento, provate a sfruttare il privilege elevation per

utilizzare il ben noto comando ping (binario con setuid settato per sfruttare i raw socket) senza privilegi speciali..

Infine possiamo considerare l'utilizzo di Systrace come IDS (Intrusion Detection System).

Grazie allo studio dei logs generati da applicazioni ed operazioni non coperte da policy, è possibile individuare intrusioni e permettere un'analisi "post mortem" (forensic) di un sistema compromesso.

Questo articolo ha voluto dare solo una visione veloce dell'utilizzo di Systrace.

Nella prossima puntata avremo l'occasione di studiare un caso reale, imparando a generare policy in modo più specifico e introducendo ad esempio l'utilizzo di un "shell wrapper"!

#### Riferimenti:

<http://www.systrace.org>

sito ufficiale

<http://www.citi.umich.edu/u/provos>

home page di Niels Provos

<http://www.blafasel.org/~floh/he>

systrace policies repository

<http://www.openbsd.org/cgi-bin/man.cgi>

man pages

<http://www.deathbyice.com/mada/systrace.vim>

systrace.vim