Usare LIDS per fortificare il proprio kernel Linux

In questa rubrica è già stato affrontato il problema di come rendere più sicuro (*trusted*) il proprio sistema operativo[1]. La soluzione corretta dovrebbe essere di progettare dall'inizio il Sistema Operativo con criteri assoluti di sicurezza. Questo però porta a tempi di sviluppo e aggiornamento molto lunghi, gestione estremamente difficile e molto ridotte possibilità di utilizzo. Quindi nella realtà di tutti i giorni si cerca al più di fare il contrario, aggiungere un po' di sicurezza al Sistema Operativo.
In questo, ed in alcuni possibili futuri articoli, considereremo Linux come un esempio alla portata di tutti.[2] Installeremo delle *patch* al Sistema Operativo che cercano di renderlo più *trusted*. Dalla installazione e configurazione di queste patch e le loro associate funzionalità, toccheremo con mano cosa voglia dire in pratica rendere più sicuro un comune odierno Sistema Operativo.

Il **Linux Intrusion Detection System - LIDS** (http://www.lids.org/), è uno dei più conosciuti progetti per rendere un kernel Linux più sicuro. La scelta del nome è anche significativa, si parla solo di *detection* mentre molte delle funzionalità sono per la *prevention*, questo evita di creare l'illusione della possibilità di raggiungere l'assoluta sicurezza. L'idea di LIDS è di trasformare il kernel di Linux dalla modalità DAC (Discretionary Access Control) comune agli OS Unix, alla modalità MAC (Mandatory Access Control). Come al solito in questa rubrica, cercheremo di capire che cosa è e come funziona LIDS, installandolo e configurandolo.

Bisogna prima di tutto se non lo si ha già, scaricare[3] un kernel da http://www.kernel.org/, poi il corrispondente patch da http://www.lids.org/. Creiamo poi le directory /usr/src/linux-2.4.19.lids e /var/lids-1.1.2rc4-2.4.19 (le versioni per il nostro esempio sono 1.1.2rc4 per LIDS e 2.4.19 per il kernel) ed in esse rispettivamente spacchettiamo con `tar xzvf` i sorgenti appena scaricati. Creiamo poi il seguente link:

```
ln -s /usr/src/linux-2.4.19.lids /usr/src/linux
```

Infine eseguiamo i comandi

```
cd /usr/src/linux-2.4.19.lids
patch -p1 < /var/lids-1.1.2rc4-2.4.19/lids-1.1.2rc4-2.4.19.patch
```

Possiamo ora configurare un kernel Linux come al solito con `make menuconfig` oppure `make xconfig`, in Tabella 1 sono riportati i principali parametri per la configurazione di LIDS.

1 *Hardening di un Sistema Operativo*, ICT Security, Novembre 2002
2 Linux è stato scelto unicamente per la disponibilità del codice sorgente e la presenza di vari progetti, incluso uno della NSA americana, che mirano a crearne versioni *trusted*.
3 Prima di tutto, controllare hash e firme digitali dei pacchetti scaricati!

Attenzione però, un kernel compilato con LIDS non supporta RAM Disk iniziali, pertanto è conveniente compilare un kernel statico, quindi senza il supporto di moduli del kernel caricabili a run-time. Inoltre, una **capability**[4] di LIDS permette di prevenire l'inserimento di moduli nel kernel in run-time una volta effettuato il *sealing* del kernel. Prima di compilare il kernel, modificare il file Makefile presente nella directory principale aggiungendo la EXTRAVERSION, ad esempio `EXTRAVERSION = .lids` , proprio all'inizio del file stesso. I seguenti comandi finiscono l'installazione del kernel

```
make dep clean bzImage
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.19.lids
cp System.map /boot/System.map-2.4.19.lids
cp .config /boot/config-2.4.19.lids
cp linux /boot/vmlinux-2.4.19.lids       [opzionale]
```

L'ultimo è opzionale. Va poi inserito il nuovo kernel nella configurazione del bootloader (lilo o grub) e nel caso di lilo anche dato il comando `/sbin/lilo -v.` In ogni caso conviene mantenere un kernel normale come secondo boot di emergenza manuale. A questo punto possiamo compilare i tools di LIDS per la configurazione delle regole:

```
cd /var/lids-1.1.2rc4-2.4.19
./configure -prefix=/usr --disable-versions-checks
make VIEW=1
mkdir -p /etc/lids/example; make install
cd lidstools; cp lidsadm.8.gz lidsconf.8.gz /usr/share/man/man8/
```

Nella versione utilizzata il comando `make install` aveva un paio di problemi, risolti con il comando `mkdir` e l'ultimo comando. Inoltre `make install` chiede di inserire una nuova password,[5] questa password verrà richiesta da LIDS per la manutenzione delle regole. E' importante inserire la password di LIDS prima di fare il reboot nel nuovo kernel, altrimenti sarà possibile fare modifiche solo con un boot di emergenza. I programmi installati sono `/sbin/lidsadm` e `/sbin/lidsconf` con le relative man page.

Inseriamo ora nel file di boot `/etc/rc.local` le seguenti istruzioni

```
if [ `uname -a | grep '.lids' | wc -c` -gt 1 ]; then
  echo "LIDS kernel, sealing..."
  /sbin/lidsadm -I
else
 echo "not a LIDS kernel, dangerous mode!!!"
fi
```

Il comando `lidsadm -I` attiva effettivamente LIDS, effettuando quello che è detto il *sealing* del kernel. Prima dell'esecuzione di questo comando LIDS applica tutte le restrizioni (ACL) per l'accesso ai file, ma per le **capability** dei programmi in esecuzione si comporta come un Intrusion

4 Il termine *sealing* verrà spiegato più avanti mentre **capability** verrà spiegato nella seconda parte di questo articolo.
5 La password si può inserire anche con il comando `lidsconf -P.`

Detection System, segnalando solo la violazione delle regole date; dopo l'esecuzione di `lidsadm -I` ogni violazione verrà impedita. Questo da la possibilità di far partire dei programmi, ad esempio server Web, DNS, SMTP ecc., prima che LIDS prenda il controllo del kernel. Infatti in alcuni casi questi server non riescono a partire se LIDS è attivo, a meno di modificare il sorgente del programma stesso. La via di uscita alternativa è di attivare prima il programma in questione e poi LIDS. Per massima sicurezza ovviamente LIDS dovrebbe essere attivato per primo, ma alla fine in pratica viene di solito attivato per ultimo, come nell'esempio dato precedentemente dove lo script `rc.local` è l'ultimo eseguito allo startup. In caso di emergenza, è possibile neutralizzare il comando `lidsadm -I` passando al kernel al momento del boot da console l'opzione `security=0`.
Prima ancora della configurazione di LIDS, è necessario fare alcune altre modifiche perché lo startup del nuovo kernel funzioni correttamente. Le modifiche più comuni sono: eliminare il comando `depmod -a` dagli script di startup, ad esempio da `/etc/rc.d/rc.sysinit`, poi sempre negli script di startup aggiungere l'opzione `-n` tutte le volte che compare il programma `mount`, ed infine sostituire il file `/etc/mtab` con un link simbolico a `/proc/mounts`. Altre modifiche potrebbero essere necessarie a seconda dei programmi installati. Queste modifiche potrebbero generare dei warning allo startup o shutdown del sistema, ma di solito questi messaggi si possono ignorare.

Passiamo ora a descrivere la sua configurazione di LIDS. Il cuore di LIDS è nei suoi file di configurazione in /`etc/lids`. Il file `lids.pw` contiene la password crittata di LIDS e si gestisce con `lidsconf -P`. Se nella configurazione del kernel si era permesso di inviare i messaggi di log via network, i relativi parametri devono essere inseriti in `lids.net`. Il file `lids.cap` contiene le definizioni globali di LIDS. La logica di LIDS è la seguente. Le restrizioni di LIDS si applicano a tutti i programmi in esecuzione come root. Ogni programma che è in esecuzione ha delle **capabilities** che gli permettono di fare o meno certe operazioni, ad esempio aprire un network socket o leggere un certo file. In `lids.cap` sono elencate più di 30 capabilities, descritte ampliamente, alcune già presenti nel kernel Linux, altre introdotte da LIDS. Il segno + o - all'inizio della riga indica se la capability è permessa o meno per tutti i programmi. Come si vede dall'estratto riportato in Tabella 2, in questa configurazione ad esempio a nessun programma è permesso di accedere a raw socket o fare un chroot, ma ovviamente nell'ultimo file di configurazione è possibile specificare regole dettagliate ed eccezioni alle regole globali.
L'ultimo file è `lids.conf`, è generato automaticamente da `lidsconf` e permette di specificare le regole per ogni file o programma. La sintassi di `lidsconf` è la seguente: l'opzione -A indica

che si vuole aggiungere una nuova regola, questa opzione è seguita dalle opzioni descritte nella Tabella 3. Riportiamo qui alcuni esempi dalla ovvia interpretazione una volta letta la Tabella 3:

```
lidsconf -A -o /sbin -j READ
lidsconf -A -o /var/log/message -j APPEND
lidsconf -A -s /etc/passwd -j DENY
lidsconf -A -s /bin/login -o /etc/passwd -j READ
lidsconf -A -s /usr/sbin/httpd -o CAP_NET_BIND_SERVICE 80\
           -i -1 -j GRANT
lidsconf -A -s /usr/sbin/httpd -d -o /var/www -j READ
lidsconf -A -s /bin/login -o /etc/shadow -t 0900:1800 -j READ
lidsconf -A -s /usr/sbin/snort -o CAP_HIDDEN -j GRANT
```

Ad esempio l'ultimo comando vuol dire che nessun utente o programma può vedere se snort è in esecuzione utilizzando ps, top o i file in /proc,[6] eccetto l'amministratore in una LidsFreeSession, come discusso più avanti. Bisogna notare inoltre che solo ad un eseguibile il cui file è protetto da una precedente regola può essere data l'autorizzazione ad esempio di scrivere su file che altrimenti sarebbero read-only.

E' consigliato crearsi un semplice shell script con tutti i comandi `lidsconf` e che cominci con `lidsconf -Z` per azzerare la configurazione esistente. Ogni volta che si fa una modifica allo script basta ri-eseguirlo per aggiornare `lids.conf`. Nella documentazione di LIDS sono riportati molti esempi ed alcuni script già pronti per dare a vari programmi, ad esempio apache o ssh, le necessarie autorizzazioni.

Finalmente, dopo aver creata la nostra configurazione, possiamo fare un reboot della macchina utilizzando il nuovo kernel. Se tutto va bene (quasi impossibile) vedremo dei messaggi di LIDS quando fa il sealing del kernel. Altrimenti molti messaggi di errore compariranno e vari programmi non riusciranno a partire. Bisogna con molta calma e pazienza annotarsi tutti i messaggi di errore, capire il loro significato, ed aggiungere i permessi mancanti ai file od agli eseguibili. Ad esempio il messaggio

```
Jan  9 20:51:58 thishost kernel: LIDS: ntpd (dev 3:6 inode 36415) pid
550 ppid 1 uid/gid (38/38) on (null tty) :  violated CAP_SYS_TIME
```

vuol dire che /usr/sbin/ntpd non ha la capability CAP_SYS_TIME che perciò va aggiunta nel file di configurazione con il comando

```
lidsconf -A -s /usr/sbin/ntpd -o CAP_SYS_TIME -j GRANT.
```

Per fare questo bisogna disabilitare almeno localmente LIDS poiché altrimenti non è possibile modificare i file di configurazione di LIDS stesso. Il comando

```
lidsadm -S -- -LIDS
```

chiede la password di LIDS e poi solo per la sessione da cui si è data la password, disabilita LIDS. In ogni istante è possibile avere una sola sessione senza LIDS (LidsFreeSession=LFS). E' anche

6 Ma il programma potrebbe essere scoperto dalle sue attività sul network o dai suoi file di log.

possibile disabilitare globalmente LIDS con l'opzione `-LIDS_GLOBAL`. L'opzione `+LIDS` chiude la LFS. Dopo aver modificato le regole, con i comandi `lidsadm -S -- +RELOAD_CONF` e `lidsconf -U` facciamo rileggere le configurazioni a LIDS. In pratica è necessario entrare in una LFS per fare la maggioranza dei lavori di manutenzione dell'amministratore, ad esempio aggiornamento di programmi ecc., e dare i comandi di aggiornamento della configurazione alla fine di ogni sessione di manutenzione. Quindi la password di LIDS diventa la vera password dell'amministratore senza la quale ben poco, se non nulla, si può fare.
Per la nostra esperienza personale, i problemi maggiori derivano dal dare le corrette autorizzazioni ai processi allo startup ed allo shutdown. Consideriamo come semplice esempio la capability CAP_SYS_RAWIO per `mount`, se non si vuole permettere di montare o smontare i dischi dopo lo startup, come è possibile allora fare uno shutdown della macchina senza disabilitare globalmente LIDS?

Il modello di sicurezza di LIDS è relativamente semplice. Ogni programma in esecuzione ha dei *permessi* che gli permettono di accedere a file od a servizi di sistema. La regolazione di questi permessi è abbastanza fine, ma ad esempio non si considera l'utente che sta eseguendo il programma, in altre parole UID, GID ecc. non sono considerati nelle regole. Per quello che non è specificato dalle regole MAC di LIDS, si applicano ancora le regole DAC normali. LIDS è pertanto utile per server, ove i programmi in esecuzione dopo lo startup sono sempre gli stessi e non vi sono utenti collegati direttamente alla macchina. L'idea è che una volta fatto il *sealing* del kernel, la macchina è effettivamente congelata, ed ogni nuovo processo non previsto è proibito. Se abbiamo ben configurato le regole, LIDS offre una ragionevole sicurezza che, anche se un programma in esecuzione permette un root exploit ed un attaccante riesce a diventare l'amministratore della macchina, senza la password di LIDS l'intruso potrà fare ben poco, neppure cancellare le proprie tracce dai file di log.

Un suggerimento finale: per ben capire quanto complicato diventi gestire una macchina *più sicura*, un ottimo esercizio è di provare ad installare LIDS su di una macchina che deve fare da server Web, DNS e SMTP. Calma, coraggio e ... buon divertimento.

In futuri articoli ci proponiamo di discutere altri modelli di sicurezza, quali i progetti Selinux dell'NSA, Openwall e RSBAC.

Andrea Pasquinucci
Consulente di Sicurezza Informatica

pasquinucci@ucci.it

Riferimenti Bibliografici:

[1]LIDS, http://www.lids.org/

[2]An Overview of LIDS, Brian Hatch, http://online.securityfocus.com/infocus/1496

[3]Per una review di aspetti di sicurezza dei sistemi operativi si veda ad esempio: *The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments,* P.A. Loscocco, S.D. Smalley, P.A. Muckelbauer, R.C. Taylor, S.J. Turner, J. F. Farrell, NSA, http://www.nsa.gov/selinux/background.html

[4]La definizione originaria di MAC è data in: *Trusted Computer System Evaluation Criteria* (TCSEC, Orange Book), DOD 5200.28-STD. Department of Defense, December 1985; una definizione più generale è in: *DTOS General System Security and Assurability Assessment Report,* Secure Computing Corporation,Technical report MD A904-93-C-4209 CDRL A011 June 1997, http://www. securecomputing.com/randt/HTML/dtos.html

[5]Per una definizione delle capability per Unix si veda il Draft POSIX 1003.1e, http://wt.xpilot.org/publications/posix.1e/download.html

[6]Selinux, http://www.nsa.gov/selinux.index.html

[7]Openwall, http://www.openwall.com/linux/

[8]RSBAC, http://www.rsbac.org/

```
Code maturity level options
  [*]   Prompt for development and/or incomplete code/drivers

General Setup
  [*]   Sysctl Support

Linux Intrusion Detection System
  [*] Linux Intrusion Detection System support  (EXPERIMENTAL)
  --- LIDS features
  (256)   Maximum protected objects to manage
  (256)   Maximum ACL subjects to manage
  (256)   Maximum ACL objects to manage
  [ ]   Hang up console when raising a security alert
  [*]   Security alert when execing unprotected programs before
          sealing
  [ ]   Do not execute unprotected programs before sealing LIDS
  [ ]   Attempt not to flood logs
  [*]   Allow switching LIDS protections
  [ ]   Restrict mode switching to specified terminals
  (3)   Number of attempts to submit password
  (3)   Time to wait after a fail (seconds)
  [ ]   Allow any program to switch LIDS protections
  [*]   Allow reloading config. file
  [ ]   Port Scanner Detector in kernel
  [ ]   Send security alerts through network
  [ ]   LIDS Debug
```

Tabella 1. Principali parametri riguardanti LIDS nella configurazione del kernel

```
### 13: - Allow use of RAW sockets
### 13: - Allow use of PACKET sockets
-13:CAP_NET_RAW

### 14: - Allow locking of shared memory segments
### 14: - Allow mlock and mlockall
+14:CAP_IPC_LOCK

### 15: Override IPC ownership checks.
+15:CAP_IPC_OWNER

### 16: Insert and remove kernel modules.
-16:CAP_SYS_MODULE

### 17: - Allow ioperm/iopl and /dev/port access
### 17: - Allow /dev/mem and /dev/kmem acess
### 17: - Allow raw block devices (/dev/[sh]d??) access
-17:CAP_SYS_RAWIO

### 18: Allow use of chroot()
-18:CAP_SYS_CHROOT

### 19: Allow ptrace() of any process
-19:CAP_SYS_PTRACE

### 20: Allow configuration of process accounting
+20:CAP_SYS_PACCT

### 25: - Allow manipulation of system clock
### 25: - Allow irix_stime on mips
### 25: - Allow setting the real-time clock
-25:CAP_SYS_TIME

### 29: Restricts viewable processes by a user.
+29:CAP_HIDDEN

### 30: Allow to kill protected processes
-30:CAP_KILL_PROTECTED

### 31: Protect process against signals
+31:CAP_PROTECTED
```

Tabella 2. Estratto di lids.cap

```
-s soggetto=/path/eseguibile (se si omette si applica a tutti
   gli eseguibili) inteso come programma in esecuzione
-o oggetto=/path/file, directory, oppure una capability
   (opzione necessaria)
-d (senza ulteriori argomenti) la directory specificata da -o è
   l'unica all'interno della quale il soggetto può agire
-t un intervallo temporale (se omesso vuol dire sempre)
-i il numero di figli (execve) di un processo che ereditano dal
   padre la regola
-j azione da applicare quando l'eseguibile (soggetto) agisce sul
   file (oggetto): DENY, READONLY, APPEND, WRITE o IGNORE
   oppure GRANT se l'oggetto è una capability (opzione necessaria)
```

Tabella 3. Opzioni di lidsconf -A