

Sistemi operativi sicuri
prof. Federico Simonetti

Hardening locale di un server basato su piattaforma Linux

Beretta Andrea <beretta_andrea@tiscali.it>

1. Scelta della distribuzione	2
2. Sicurezza fisica	4
3. Sicurezza a livello di filesystem	8
4. Integrita' del filesystem	22
5. Utenti, gruppi e restrizione dei privilegi locali	32
6. Pluggable Authentication Modules	40
7. Sistema di logging	47
8. Backup	61
9. Kernel	68
10. Bibliografia	78

Scelta della distribuzione

Prima di iniziare a considerare tutte le operazioni di partizionamento, e configurazione successive all'installazione, e' necessario ragionare sulla scelta della distribuzione da adottare.

Esiste infatti una grandissima varieta' di distribuzioni Linux, ognuna con le sue caratteristiche piu' o meno peculiari. Per un sistema desktop puo' essere naturale scegliere la distribuzione piu' user-friendly, o in generale quella che piu' soddisfa i propri gusti personali, ma per un server la scelta e' decisamente piu' complessa. Bisogna infatti prestare un occhio particolare alla sicurezza ed alla stabilita' del sistema oltre che del software che si va ad installare. In generale alcune caratteristiche da valutare attentamente nella scelta possono essere le seguenti :

- **Quantita' di software installato:** sarebbe opportuno scegliere una distribuzione che installi la minore quantita' possibile di software. L'ideale sarebbe avere un sistema "minimale" una volta completata l'installazione, al quale aggiungere poi solo i pacchetti necessari. Oltre che inutile, la presenza di pacchetti "superflui" e' anche potenzialmente dannosa, poiche' essi potrebbero esporre ad ulteriori vulnerabilita', oltre a quelle legate ai software strettamente necessari. In questa ottica, e' bene tenere a mente la regola d'oro : " **Quello che non c'e' non si puo' rompere**".
Naturalmente nulla vieta di rimuovere in seguito il software inutile, o di eseguire una scelta attenta in fase di installazione, ma personalmente trovo piu' logico partire con un sistema "pulito" al quale aggiungere le componenti necessarie a soddisfare le esigenze previste.
- **Disponibilita' di aggiornamenti:** essendo la sicurezza e la stabilita' del software un requisito fondamentale, devono essere evitate assolutamente distribuzioni che non rilascino con sufficiente rapidita' hotfix ed updates dei pacchetti critici da questo punto di vista.
- **Documentazione e supporto:** una distribuzione priva di documentazione o pressoché sconosciuta ai piu' potrebbe non essere una scelta particolarmente oculata. Avere una buona documentazione, ed una community attiva alle spalle sono requisiti fondamentali. La mancanza di guide, mailing list o newsgroup di riferimento renderebbero estremamente complessa la risoluzione dei problemi che potrebbero verificarsi.
- **Un buon sistema di pacchettizzazione:** un tool che consenta di gestire in maniera efficace tutti i pacchetti e le loro dipendenze non puo' mancare. Gestire il tutto in maniera completamente manuale sarebbe un caos pazzesco.
- **Longevita':** essendo interessati principalmente alla sicurezza ed alla stabilita' del sistema devono essere accuratamente evitate distribuzioni che rilascino nuove versioni troppo frequentemente. Molto spesso distribuzioni che adottano questo approccio tendono ad includere le ultime versioni disponibili dei diversi software, anche se esse non sono sufficientemente testate e stabili. Oltre al dubbio che release così frequenti, non consentano un debito controllo del software presente, bisogna considerare che un sistema così "mutevole" richiederebbe spesso aggiornamenti sostanziali.

Restando tra le distribuzioni piu' note ritengo che **Mandrake** sia da scartare a priori. Sebbene **urpmi** sia un buon sistema di gestione dei pacchetti, la distribuzione non gode di una grande reputazione in fatto di stabilita', ed essendo orientata maggiormente ad un ambiente desktop tende a preferire le ultime release disponibili dei software, anche se non sufficientemente stabili.

Un discorso simile potrebbe essere fatto anche per **Fedora**.

Distribuzioni che invece godono di una buona fama in fatto di sicurezza, con una buona disponibilita' di software, e piu' orientate alla stabilita' anziche' ad un ambiente user-friendly per l'utente potrebbero essere **SlackWare** e **Gentoo**. Nella mia scelta le ho scartate entrambe a causa del sistema di gestione dei pacchetti. In realta' **emerge** sarebbe un sistema molto potente per l'installazione e la risoluzione delle dipendenze. Ma la necessita' di compilare tutto il software che si decide di installare rende il tutto troppo macchinoso. Volendo e' possibile usare anche in **Gentoo** software precompilato, ma essendo la "compilazione totale" la sua caratteristica principale, mi sembrerebbe un controsenso.

Alla luce di queste considerazioni, quindi, la mia scelta e' stata per **Debian**, poiche' essa soddisfa tutti i requisiti preelencati relativi all'accuratezza con cui vengono selezionati i pacchetti, gli aggiornamenti e la longevita', unita ad un ottimo sistema di gestione dei pacchetti quale **apt**.

In molte realta' lavorative pero', l'adozione di distribuzioni quali **Debian** (o comunque anche **Slackware** o **Gentoo**), potrebbe essere osteggiata, poiche' esse hanno si' alle spalle una comunita' molto nutrita e competente, ma non fanno riferimento ad alcuna organizzazione commerciale, al contrario di **Suse** o **RedHat** ad esempio. Nella maggior parte dei casi, dubito fortemente che il management se la sentirebbe di puntare su una distribuzione senza un supporto "ufficiale".

Trattandosi pero' di un progetto d'esame, ed avendo quindi un minimo di liberta', ho optato per la distribuzione che ritenevo piu' corretta e con la quale ho maggiore dimestichezza , tralasciando gli aspetti piu' "commerciali", sebbene sappia quanto peso essi possano avere in certi contesti lavorativi.

NOTA: Trattandosi di una distribuzione a pagamento non ho avuto modo di provare le ultime versione di **RedHat**, quindi non l'ho considerate nella scelta. Non ho considerato neppure **Suse**, poiche' non ho mai potuto provarla seriamente, aldila' di alcune occasioni saltuarie circa tre anni fa.

Sicurezza fisica

Prima ancora di entrare nel dettaglio degli aspetti di configurazione del sistema, e' fondamentale fare anche alcune considerazioni sulla sicurezza fisica dell'ambiente in cui la macchina che si deve preparare andra' a trovarsi.

Nel caso del computer usato come esempio per questo progetto, la sicurezza fisica non e' un'aspetto fondamentale, trovandosi la macchina in un ambiente "casalingo" e privo di utenti sufficientemente smalzati. Il discorso cambia invece radicalmente nel caso di macchine in un ambiente di lavoro, dove le persone che potrebbero accedervi sono piu' numerose e variegate.

Quindi, potrebbe essere un buon punto di partenza iniziare a ragionare su chi sono le persone che hanno fisicamente accesso alle macchine e se sia un loro diritto o meno accedervi.

Come gia' detto, prima ancora di considerare la sicurezza relativa alla macchina, e' bene ragionare anche su alcuni requisiti che potrebbero essere necessari per l'ambiente che ospita il sistema od i sistemi oggetto dell'attivita' di hardening. Naturalmente ogni caso deve essere valutato singolarmente, poiche' possono essere diverse le condizioni in cui si puo' ritrovare. Se possibile, pero', alcune linee da seguire potrebbero essere le seguenti :

- Accesso limitato e controllato al locale dove risiedono le macchine che si vogliono proteggere. Se il budget lo consentisse, potrebbe essere una soluzione l'adozione di qualche sistema di autenticazione, piu' o meno forte a seconda della criticita' delle risorse da proteggere.
- Climatizzazione della stanza. La presenza di numerose macchine in funzione all'interno di un locale (od a maggior ragione in un armadio rack), causerebbe un innalzamento della temperatura, che potrebbe portare a danneggiamenti delle macchine stesse. E' fondamentale quindi prevedere un sistema di climatizzazione per mantenere una temperatura adeguata nell'ambiente.
- Proteggere i sistemi anche da eventuali atti di vandalismo, sabotaggio, furto, allagamento ecc. In questo senso potrebbe essere una buona idea l'adozione di armadi rack. Basta immaginare, ad esempio, un addetto alle pulizie (e quindi autorizzato ad accedere alla sala macchine) che sbadatamente stacchi un cavo di alimentazione o rovesci un secchio nei pressi di una macchina.
- Utilizzo di gruppi di continuita' o gruppi elettrogeni, per consentire di continuare ad erogare elettricita' anche in caso di problemi alla linea principale.
- Previsione di un piano anti-incendio adeguato. Ad esempio, estintori idrici, a polvere o ad anidride carbonica potrebbero causare diversi danni all'hardware. Essendo gli estintori ad Halon vietati per la tossicita' degli idrocarburi alogenati che li costituiscono, si potrebbe pensare di adottare estintori a **NAF**, magari di tipo automatico.

Dopo questi presupposti si puo' passare ad alcune considerazioni sulla sicurezza della singola macchina. Partendo dal livello piu' basso, bisogna considerare il **BIOS**, in modo da configurarlo al meglio.

Prima di tutto e' necessario impostare una password a protezione del **BIOS** stesso. In caso contrario, infatti, chiunque possa avere accesso fisico alla macchina, potrebbe accedere al **BIOS** prima del caricamento del bootloader e ad esempio modificare la sequenza di boot, facendo avviare il sistema da un floppy o da un cd. In questo modo e' sufficiente una qualsiasi distribuzione live, per ottenere accesso con privilegi da amministratore al contenuto del disco rigido.

Volendo, e' possibile impostare anche una password all'avvio del pc, ma bisogna tenere presente, che in caso di reboot della macchina, essa non potra' ripartire automaticamente, poiche' prima del boot del sistema e' necessario inserire tale password. D'altro canto essa fornisce un segno inequivocabile che la macchina e' stata riavviata, sia che si sia trattato di un riavvio "voluto" o di una mancanza di alimentazione.

E' inoltre caldamente consigliato modificare la sequenza di boot predefinita, disabilitando il boot da floppy e da cdrom. In diversi **BIOS** infatti, la sequenza di boot predefinita prevede che vengano prima caricati floppy o cdrom, anziche' l'hard disk. In questo caso impostare una password per l'accesso al **BIOS**, lasciando queste impostazioni di

default sarebbe poco efficace, poiche' sarebbe ancora possibile eseguire il boot da floppy o da cdrom.

Nel caso dovesse presentarsi la necessita' di avviare la macchina con una distribuzione live, ad esempio a causa di problemi del sistema operativo installato, sara' sufficiente accedere nuovamente al **BIOS**, inserire la password precedentemente impostata e modificare nuovamente la sequenza di boot, a seconda delle proprie esigenze.

Un ulteriore passo da eseguire, e' la modifica delle impostazioni iniziale del bootloader. I due bootlader piu' noti in ambiente Linux sono **LILO** e **Grub**. Le diverse considerazioni saranno relative a **LILO**, ma possono essere applicata anche a **Grub**. Essenzialmente le differenze saranno nei file di configurazione o nel modo per passare parametri all'avvio dell'immagine.

E' infatti possibile passare dei parametri al kernel, specificandoli appunto tramite l'interfaccia del bootloader. Ad esempio appendendo al nome dell'immagine la stringa **init=/bin/bash**, e' possibile accedere ad una shell (nel caso la **bash**) con permessi da utente root, e senza alcuna autenticazione.

Un altro esempio e' il parametro **single**, che consente di avviare il sistema in modalita' single user, ottenendo ancora una shell con permessi da utente root. Per la precisione ho avuto modo di notare che la cosa vale per le distribuzioni **Mandrake**, **Fedora** e **Gentoo**, mentre in **Debian** viene comunque richiesta la password dell'utente root per accedere alla shell (e' infatti presente in **/etc/inittab** la riga `~:S:wait:/sbin/sulogin` per chiedere la password di root anche in caso di avvio in modalita' single user)

La falla dal punto di vista della sicurezza, risiede nel fatto che nella maggioranza delle distribuzioni il bootloader viene installato con una configurazione di default, che consente di passare parametri al kernel liberamente.

Si rende quindi necessario intervenire sul file di configurazione del bootloader (**/etc/lilo.conf** nel caso di **LILO**), in modo da aumentarne la sicurezza.

LILO prevede due opzioni che possono tornare utili allo scopo: **password** e **restricted**.

La direttiva **password** consente di bloccare il boot dell'immagine finche' non viene inserita un'opportuna parola chiave, specificata nel file di configurazione. **Restricted** invece, fa in modo che il bootloader chieda l'inserimento di una password solo se si cerca di passare qualche parametro al kernel. In caso contrario l'avvio puo' avvenire regolarmente.

Poiche' tali password verranno scritte all'interno del file **/etc/lilo.conf** e' importante verificare i permessi ad esso relativi, ed in caso di necessita' impostarli in modo da consentire accesso in lettura e scrittura esclusivamente all'utente root.

Sempre relativamente al bootloader, potrebbe essere una buona idea anche prevedere due differenti "etichette". Una per il normale avvio del server, ed una seconda per eventuali operazioni di manutenzione, avviando il sistema con un determinato runlevel (ad esempio 4) dove sono stati disattivati tutti i servizi non necessari, ed impostate regole di filtraggio del firewall particolarmente restrittive.

Per la precisione il **lilo.conf** della macchina preparata nel corso del progetto e' stato il seguente:

```
lba32
boot=/dev/hda
root=/dev/hda5
install=/boot/boot-menu.b
map=/boot/map
delay=20
timeout=50
prompt
vga=normal
default=linux

#Linux
image=/vmlinuz
```

```

label=linux
password=P5ych0t1Cwaltz
restricted
read-only
append="rootflags=data=journal"

#manutenzione
image=/vmlinuz
label=manutenzione
password=P5ych0t1Cwaltz
restricted
read-only
#avvia con runlevel 4
append="init 4"

#Freebsd
other=/dev/hda1
label=freebsd

```

NOTA: Idealmente non dovrebbe essere presente la label per l'avvio di un ulteriore sistema operativo. In realta' la macchina che ho usato per i test aveva gia' installato un sistema freebsd che utilizzo saltuariamente, e non ho voluto rimuovere dal disco. Ho lasciato indicata anche questa immagine nel file di configurazione, in modo che si capisca perche' le partizioni del sistema Linux partissero /dev/hda5 e non da hda1.

Nel caso appena visto si e' deciso di utilizzare la direttiva **password** solamente nel caso si cerchi di passare dei parametri al kernel, e non per il semplice avvio, poiche' e' gia' stata impostata una password nel **BIOS**, per l'avvio della macchina. La scelta di obbligare all'inserimento di una password per l'avvio della macchina, e' stata fatta a scopo puramente dimostrativo. Essa e' in realta' una decisione da considerare con estrema attenzione, poiche' ad esempio il server non potra' ripartire in maniera autonoma in caso di una sospensione e successivo ritorno dell'alimentazione.

Un ulteriore operazione da eseguire dovrebbe essere disabilitare il riavvio del sistema tramite la combinazione dei tasti CTRL+ALT+DELETE . In questo modo possiamo evitare che chiunque sia in grado di accedere alla tastiera della macchina possa riavviare il sistema, e eseguire il boot da un unita' rimovibile o passare parametri al kernel tramite il bootloader. Prendendo le opportune precauzioni in fatto di sicurezza fisica e di configurazione del bootloader non si dovrebbe essere troppo esposti a questi rischi, ma la prudenza non e' mai troppa.

Per eseguire l'operazione e' sufficiente commentare in **/etc/inittab** la riga relativa alla suddetta combinazione di tasti:

```

# What to do when CTRL-ALT-DEL is pressed.
#ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

```

Sempre per impedire un riavvio brutale della macchina si potrebbe anche pensare di scollegare il cavo collegato al tasto di reset. In ogni caso si ricorda la possibilita' di mantenere le macchine all'interno di armadi rack.

Un'ultima considerazione da fare, e' la necessita' di bloccare, la postazione di lavoro quando ci si allontana da essa, sia che si acceda direttamente al server o da remoto con sistemi quali **ssh**.

Non potendo avere l'assoluta certezza che personale non autorizzato possa accedere alla postazione di lavoro, sarebbe opportuno "bloccarla" in modo che non possa venire manomessa, o che si possa scoprire nel dettaglio cosa

si stava facendo.

Due software adatti allo scopo possono essere **vlock** ed **xlock** nel caso di sessioni X. In ogni caso poiche' si sta ragionando su una macchina con funzionalita' di server andrebbe evitata l'installazione del servente grafico, poiche' non strettamente necessario.

Sicurezza a livello di filesystem

Il primo problema da valutare e' la scelta del filesystem da utilizzare ed il partizionamento dei dischi. Sebbene sia piuttosto semplice e robusto e' stato scartato **ext2**, quindi la scelta e' stata ristretta ai piu' diffusi filesystem journaled disponibili, ovvero **ext3**, **JFS**, **ReiserFs** ed **XFS**.

Perche' la scelta di scartare a priori **ext2**? Un filesystem al suo interno mantiene non solo i file veri e propri, ma anche altri dati, i metadata, che registrano diverse informazioni relative ai file: grandezza, collocazione fisica sul supporto, eccetera. Ogni volta che si crea, modifica o cancella un file il sistema deve aggiornare i metadata, per mantenerli allineati ai dati immagazzinati nei file.

Il problema è che questo implica una certa quantità di accessi fisici al disco. Se ad esempio, durante un crash del sistema, od un'interruzione dell'alimentazione il sistema stava scrivendo qualcosa sul disco, è possibile che una di queste operazioni sia stata interrotta a metà. Di conseguenza si puo' verificare un disallineamento tra quanto è stato scritto su disco ed i metadata che dovrebbero localizzarlo. Il sistema operativo allora deve controllare tutto il disco alla ricerca di questi disallineamenti, e tentare di correggerli.

Utilizzando un fs non journaled come **ext2**, in situazioni del genere, la sequenza di boot veniva tipicamente bloccata, ed eseguito un **fsck** per controllare i dischi che erano montati al momento dell'interruzione. Tale operazione non era particolarmente rapida, e a seconda delle dimensioni dei dischi richiedeva diversi minuti. Oltre alla lentezza un altro aspetto peggiore doveva essere temuto: in alcuni casi il disallineamento poteva semplicemente essere troppo grave per poter essere corretto, oppure potevano essere andati persi alcuni parametri essenziali al ripristino, e il filesystem non era in grado di trovarli da solo. Di conseguenza i dati che si stavano scrivendo venivano persi.

Per non rischiare di incorrere in simili problemi, la scelta di un filesystem di tipo journaled.

L'idea alla base e' simile alla gestione delle transazioni in un database: in pratica si mantiene all'interno della partizione un file speciale che agisce da log, dove si registrano tutti i cambiamenti che vengono fatti. In caso di crash il filesystem puo' essere riparato rapidamente, poiche' il sistema operativo leggendo il log avra' la possibilita' di completare le operazioni non correttamente completate.

A seconda dei filesystem e' possibile fare il journaling dei soli metadata (**ReiserFS** antecedenti alla versione 4, **XFS** e **JFS**), oppure completo sia di dati che metadata (come ad esempio in **ext3** e nella versione 4 di **ReiserFS**), sebbene con prestazioni inferiori rispetto al caso precedente.

XFS e' un filesystem particolarmente indicato dove si debbano trattare file di grandi dimensioni, quindi dove vi siano applicazioni per animazione tridimensionale o video editing ad esempio. Ipotizzando che la macchina che si andra' a preparare sia utilizzata come server web o mailserver, un filesystem pensato per gestire al meglio file di grandi dimensioni potrebbe non essere la scelta migliore, anche in termini di spreco dello spazio su disco.

Viene scartato anche **JFS** poiche' non considerato sufficientemente testato, fosse solo per il fatto che esso non e' incluso nativamente nel ramo 2.4 del kernel (**NOTA**: nella versione 2.4.18 esso non era presente, mentre lo e' nella 2.4.27 presente nei repository di **apt**, quindi deve essere stato inserito nel frattempo).

ReiserFS ha il vantaggio di avere prestazioni ottime, soprattutto con i files di piccole dimensioni, pero' non supporta il full-journaling, se non dalla versione 4.

Poiche', piu' che alle prestazioni, l'interesse in questo lavoro e' rivolto alla sicurezza del sistema, si predilige la scelta di un filesystem full-journaled, cercando di avere le massime garanzie in fatto di integrita' dei dati in caso di crash.

Poiche' la versione 4 di **ReiserFS** e' stata rilasciata nell'anno 2004 potrebbe non essere ancora abbastanza matura per un utilizzo su un server. Esso inoltre non e' ancora presente nel kernel 2.4.24, quindi la scelta del filesystem da utilizzare e' caduta su **ext3**, che nonostante prestazioni inferiori, offre il supporto al full-journaling ed e' gia' incluso nel ramo 2.4 del kernel.

La scelta di un filesystem opportuno non e' pero' sufficiente per ottenere una sicurezza di base del sistema quantomeno decente. Ad esempio, installare linux su di un'unica partizione, non sarebbe affatto una buona soluzione, quando sarebbe possibile separare i diversi filesystem su piu' partizioni, sfruttando magari alcune delle diverse

opzioni del comando **mount**.

Alcune linee guida da tenere presenti per un corretto partizionamento potrebbero essere le seguenti :

- Porre le directory scrivibili pubblicamente, come ad esempio **/tmp** o **/var/tmp** in partizioni separate dal resto del sistema. Basti pensare al caso di un sistema con un'unica partizione, dove gli utenti potrebbero riempire la directory **/tmp** fino ad esaurire tutto lo spazio disponibile su disco.
- Lo stesso discorso puo' valere anche per **/var/log**, dove ad esempio potrebbe essere causato un DoS facendo in modo che il sistema inizi a generare un quantitativo enorme di entries nei log di sistema fino all'esaurizione delle risorse
- Impedire l'esecuzione di programmi e script su alcuni filesystem, tramite l'opzione **noexec**. Per esempio si potrebbe decidere di non consentire l'esecuzione di programmi su directory condivise in rete, e unita' rimovibili quali dischi floppy, cdrom, pendrive. Potrebbe essere una buona scelta impedirne l'esecuzione anche nelle directory **/home** e **/var/tmp**.
Potrebbe invece esserci la necessita' di mantenere la directory **/tmp** senza l'opzione **noexec**, poiche' spesso al momento dell'installazione od aggiornamento di software pacchettizzato debbono essere eseguiti alcuni script che utilizzano appunto questa cartella. Un'alternativa e' utilizzare comunque l'opzione **noexec** nel montaggio della directory **/tmp**, e preparare uno shellscript per smontarla e rimontarla senza l'opzione **noexec**, da eseguire ad esempio prima di un aggiornamento del software.
- Impedire l'uso dei bit **SUID/SGID** nelle directory dove sono presenti i diversi eseguibili. Ad esempio un **vi** settato per errore con il bit **SUID** attivo avrebbe conseguenze catastrofiche, poiche' ogni volta verrebbe eseguito con i permessi dell'utente root. In questo modo qualsiasi utente potrebbe scrivere in luoghi dove potrebbe farlo solo l'amministratore di sistema. In generale e' sconsigliabile l'utilizzo dei bit **SUID/SGID** ove non strettamente necessario. Piuttosto se esiste la reale necessita' di eseguire un programma con permessi particolari e' meglio utilizzare il comando **sudo**.
- Evitare l'utilizzo dell'opzione **noatime**, che evita la registrazione del tempo di accesso ad un file. L'utilizzo di questa opzione consentirebbe una maggiore velocita' nelle operazioni su disco, ma a fini di controllo e verifica e' sicuramente utile avere la possibilita' di sapere a quando risale l'ultimo accesso ad un file.
- Non utilizzare l'opzione **user** nel montaggio delle unita' rimovibili quali cdrom e dischetti. In questo modo gli utenti normali non saranno autorizzati a montare tali unita'.
- Utilizzare l'opzione **noauto** per il montaggio di unita' rimovibili, in modo che esse non vengano montate automaticamente durante la fase di boot.
- Assegnare una partizione separata anche alla directory **/usr/**, e montarla in modalita' sola lettura (opzione **ro**). In caso di necessita' per un update e' possibile modificare lo shellscript precedentemente menzionato, facendo in modo che la directory sia smontata e rimontata senza l'opzione di sola lettura.

Un possibile **/etc/fstab** che implementi questo schema di partizionamento potrebbe quindi essere il successivo (**nota:** per motivi di impaginazione mancano i valori per dump e pass, che sarebbero comunque sempre settati a 0. Unica eccezione il valore 1 di pass per la partizione montata sotto /):

```
/dev/hda5      /          ext3          data=journal,defaults
/dev/hda9      none       swap          sw
proc          /proc      proc          defaults
/dev/fd0       /floppy    auto          noauto,nosuid,noexec,nodev
/dev/hdc       /cdrom     iso9660       ro,noauto,nosuid,noexec,nodev
/dev/hda6      /var       ext3
defaults,data=journal,nosuid,nodev,noexec,usrquota,grpquota
/dev/hda7      /var/tmp   ext3
```

```

defaults,data=journal,nosuid,nodev,noexec,usrquota,grpquota
/dev/hda8          /var/log      ext3
defaults,data=journal,nosuid,nodev,noexec
/dev/hda9          /tmp          ext3
defaults,data=journal,nosuid,nodev,noexec
/dev/hda10         /usr          ext3          defaults,data=journal,ro,nodev
/dev/hda11         /home         ext3          defaults,data=journal,nosuid,
noexec, noexec, usrquota, grpquota

```

Come si puo' notare tutte le partizioni del disco fisso, ad eccezione di proc e swap, vengono montate con l'opzione **data=journal**. Tale opzione consente di montare il filesystem ext3 in modalita' full-journaling. Per potere montare anche / in questa modalita' e' necessario anche passare **rootflags=data=journal** come parametro al kernel, tramite il bootloader. Anziche' specificarlo ogni volta, si e' inserito tale parametro direttamente in **/etc/lilo.conf**, utilizzando la direttiva **append**.

Contrariamente a quanto si potrebbe pensare, inoltre, in molti casi il full journaling si e' dimostrato estremamente piu' veloce rispetto agli altri tipi di journaling, sebbene si pensasse dovesse essere il piu' lento dei tre. Dopo alcune discussioni sulla mailing list del Kernel Linux, Andrew Morton ha eseguito alcuni test per verificare la cosa. Egli ha predisposto uno script per cercare di scrivere quantita' di dati il piu' velocemente possibile su un filesystem, e durante tale scrittura, ha cercato di leggere files con dimensione di 16Mb, sempre dallo stesso filesystem. I risultati sono stati stupefacenti, poiche' **ext3** con l'opzione **data=journal** avrebbe consentito di leggere da 9 a 13 volte piu' velocemente tali files rispetto alle altre modalita' sempre per ext3, ma anche rispetto a **ReiserFs** ed addirittura **ext2**, che non ha alcun overhead dovuto al journaling.

Sembrerebbe quindi, che in server particolarmente "carichi", dove si presenti sovente la necessita' di leggere e scrivere contemporaneamente sullo stesso filesystem, **ext3** con l'opzione di full-journaling, oltre a garantire una maggiore sicurezza consente anche prestazioni superiori (fonte: <http://www-128.ibm.com/developerworks/linux/library/l-fs8.html#4>).

Ritornando alle partizioni, quelle montate sotto **/var**, **/var/tmp** e **/home** presentano anche le opzioni **usrquota** e **grpquota**, relative appunto alle quote disco.

L'utilizzo delle quote disco consente di limitare lo spazio ed il numero di files che un utente puo' avere su uno specifico dispositivo. Ai fini della sicurezza, imporre dei limiti e' sicuramente utile, ad esempio per impedire che un utente possa riempire tutto lo spazio assegnato ad **/home**, creando quindi disagi a chi altro avesse bisogno di salvare dei dati nella propria home directory.

La scelta del filesystem **ext3** e' stata fatta, tra gli altri motivi, anche per il fatto che esso supporta "nativamente" le quote disco, senza necessita' di applicare patch al kernel, come invece la versione 3 di **ReiserFs**.

Esse consentono inoltre una certa flessibilita' nella definizione dei limiti, con le cosiddette soft e hard quota. Al raggiungimento della quota soft il sistema lascia all'utente un determinato periodo di tempo, definito grace-time, per rientrare nei limiti stabiliti. Allo scadere di tale periodo l'utente non potra' piu' fare nulla se non cancellare dati. In questo lasso di tempo l'utente puo' ancora lavorare normalmente e scrivere sul dispositivo, sempre rimanendo entro la hard quota.

Al raggiungimento della quota hard, invece, non viene concesso alcun periodo di grazia ed all'utente viene semplicemente consentito di cancellare dei file per tornare a rispettare i limiti di spazio.

L'utilizzo di due quote diverse garantisce un minimo di flessibilita' e di operativita' agli utenti, a seconda del grace time consentito e della differenza tra le due quote.

Naturalmente le due quote possono essere fissate oltre che per lo spazio su disco, anche per il numero di file che un'utente puo' possedere.

Parallelamente alle user quota esistono anche le group quota, concettualmente identiche, ma pensate per imporre dei limiti per gruppo anziche' per utente.

La procedura seguita per l'impostazione delle quote utente e' stata la seguente:

1. Installare il pacchetto **quota**, con il comando **apt-get install quota**
2. Modificare il file **/etc/fstab** aggiungendo le opzioni **usrquota** e **grpquota** alle partizioni in cui si vogliono utilizzare le quote disco.
3. Riavvio del sistema, in modo da "attivare" il sistema di quote, senza dovere smontare e rimontare manualmente le diverse partizioni e creare nella root di ognuna di esse i files **quota.user** e **quota.group**.
4. Verificare che in fase di boot venga eseguito **/etc/init.d/quotarpc** . Normalmente gli script di avvio del sistema vengono automaticamente modificati durante l'installazione del pacchetto.
5. Impostare le quote disco per i diversi utenti utilizzando il comando **edquota -u** . Allo stesso modo per i gruppi si deve utilizzare **edquota -g** . Ad esempio con il comando **edquota -u standard**, si potranno impostare le quote per l'utente standard, tramite l'editor di sistema predefinito :

Filesystem	blocks	soft	hard	inodes	soft	hard
		[...]				
/dev/hda11	16	40	60	4	1024	1024
		[...]				

Ad esempio l'utente per cui si stanno editando le quote avra' un limite soft di 40 Mb ed hard di 60 Mb. Viene inoltre impostato un limite di 1024 inodes all'utente. Naturalmente questi valori debbono riflettere le effettive necessita' degli utenti del sistema. L'ideale sarebbe potere osservare in media quanto spazio e quanti inodes vengono utilizzati da diversi utenti (ad esempio ancora tramite **edquota -u**), e da li' creare delle quote ad hoc.

Quindi, non imporre alcun limite sugli inodes sarebbe un comportamento da evitare, poiche' un filesystem non e' in grado gestire un numero infinito di files. Senza impostare alcuna quota un utente potrebbe infatti preparare script o programmi malevoli che creino volutamente un quantitativo sproporzionato di files. Teoricamente essendo abilitata l'opzione **noexec** sulle partizioni dove un utente comune ha la possibilita' di scrittura, non dovrebbe essere consentita l'esecuzione di script in queste posizioni, ma e' sempre meglio essere un po' paranoici.

6. Impostare la durata del grace-period per le diverse partizioni con **edquota -t**.
7. Inserire il programma **warnquota** all'interno di un cronjob, in modo da eseguirlo con una determinata frequenza. Il programma in questione verifica le quote disco per ogni filesystem sulle quali sono state attivate, ed invia un'email per avvisare gli utenti che hanno superato i limiti stabiliti. Installando il pacchetto **quota**, **apt** ha provveduto anche ad inserire uno shellscript preposto a tale scopo, chiamato appunto **quota**, nella directory **/etc/cron.daily**.

Un'altra possibilita' da considerare riguardo alla sicurezza del filesystem potrebbe essere l'uso delle **Access Control List** oltre al classico sistema di permessi utente/gruppo/altri.

In molti casi il meccanismo classico e' piu' che sufficiente, ma se dovesse esistere l'effettiva necessita' di impostare i permessi con una granularita' maggiore, l'utilizzo delle **ACL** potrebbe essere la scelta corretta.

Se il server dovesse essere mantenuto da un unico amministratore, e gli utenti non avessero esigenze particolari in fatto di permessi potrebbe essere possibile cavarsela egregiamente con il classico sistema di permessi, ma potrebbero essere diversi i casi in cui le **ACL** potrebbero tornare molto utili.

Si puo' ad esempio immaginare un fileserver, dove nella directory **/home/andrea/** si trovino il sito ed alcuni documenti dell'utente **andrea**. Owner e gruppo di tale directory, di eventuali sottodirectory e dei files ivi contenenti e' appunto **andrea**.

Ad esempio, le politiche di sicurezza potrebbero imporre che nessuno debba potere leggere o scrivere in tale cartella

oltre al proprietario (**andrea**) e l'utente **davide**.

In questo caso si potrebbero togliere i privilegi di accesso ad others (**chmod o-rwx**) ed impostare gli opportuni permessi per il gruppo associato ai files od alla directory, al quale si provvedera' ad aggiungere anche l'utente **davide**.

Questa soluzione non e' pero' detto sia quale ottimale: ad esempio gli utenti del gruppo **andrea** potrebbero essere autorizzati ad accedere, in altre porzioni del filesystem, (o ad eseguire alcuni programmi) a cui non dovrebbero invece avere accesso altri utenti aggiunti al gruppo solo per garantire loro l'accesso ai dati contenuti in **/home/andrea**, come appunto **davide**.

Una possibile soluzione potrebbe essere cambiare il gruppo associato ai files ed alle cartelle desiderati, tramite il comando **chgrp**.

In alcuni casi quindi, giocando con i gruppi e' possibile impostare dei permessi adeguati alle proprie necessita', sebbene il sistema possa risultare un po' macchinoso. Essi non sono pero' adatti se la granularita' che si richiede nella definizione dei privilegi e' maggiore.

Ad esempio si potrebbe desiderare che l'utente **davide** possa accedere con permessi **rwx** ad alcuni files, gli utenti **riccardo** ed **edoardo** con i soli permessi in lettura e l'utente **rossana** non debba accedervi in alcun modo.

In una simile situazione non potremmo consentire i permessi in lettura ad others, perche' in questo caso anche l'utente **rossana** potrebbe accedere ai files che non vogliamo possa visualizzare.

Si potrebbe ad esempio togliere il permesso di esecuzione per other alla cartella **/home/andrea**, ed aggiungere gli utenti che devono accedere ai dati ad un gruppo creato ad hoc. In questo modo gli utenti diversi dal proprietario e non appartenenti a tale gruppo non potrebbero "attraversare" la directory **/home/andrea**, e quindi accedere ai files.

Come gia' detto, pero', la granularita' ottenuta potrebbe non essere sufficiente: se si impostassero i permessi per il gruppo consentendo sia lettura che scrittura, gli utenti **riccardo** ed **edoardo** si troverebbero anche a potere scrivere su cio' che invece dovrebbero solo leggere.

Se si impostassero, invece, permessi di sola lettura ci si ritroverebbe con l'utente **davide** impossibilitato a scrivere. In un caso simile le ACL sarebbero senz'altro la soluzione piu' opportuna da adottare.

Lo stesso problema oltre che per gli accessi in lettura e scrittura ai files, potrebbe presentarsi anche per l'esecuzione dei programmi. Ad esempio le politiche potrebbero richiedere che il compilatore C, per ovvi motivi di sicurezza possa essere utilizzato solo da utenti specifici, ad esempio **andrea** e **davide**. Se si vanno a vedere i permessi con un **ls -l /usr/bin/gcc-3.3** si ottiene il seguente risultato :

```
-rwx-r-xr-x 1      root  root
```

Quindi proprietario e gruppo del file e' **root**, mentre gli altri utenti hanno su di esso permesso di lettura ed esecuzione. Non volendo consentire a tutti l'utilizzo del compilatore si potrebbe pensare di rimuovere i permessi per gli altri utenti con **chmod o-rx /usr/bin/gcc-3.3** ed aggiungere gli utenti **andrea** e **davide** al gruppo **root**.

L'aggiunta al gruppo **root** consentirebbe pero' privilegi eccessivi su altri programmi e files agli utenti **andrea** e **davide**. Anche in questo caso sarebbe possibile cambiare il gruppo associato al file, ma personalmente mi pare un meccanismo piu' macchinoso rispetto all'uso delle **ACL**. Utilizzando le Access Control List, potremmo invece definire che oltre a **root** ed agli appartenenti al gruppo omonimo solo **andrea** e **davide** possano eseguire il compilatore **gcc**:

```
setfacl -m u:andrea:rx  /usr/bin/gcc-3.3
setfacl -m u:davide:rx  /usr/bin/gcc-3.3
```

In questo modo i permessi del file diventerebbero i seguenti, consentendo l'esecuzione ad **andrea** e **davide** come si puo' verificare con il comando **getfacl /usr/bin/gcc-3.3**:

```
# file: usr/bin/gcc-3.3
# owner: root
# group: root
user : : rwx
user :andrea:r-x
user :davide:r-x
group : :r-x
other : :---
```

Ho scelto di usare **gcc** come esempio di comando perche' mi sembra particolarmente sensato consentirne l'esecuzione solo ad alcuni utenti. Personalmente, pero', eviterei di installare qualsiasi compilatore sul server, quando possibile, per limitare il rischio che qualcuno possa compilare qualche exploit, o a compromissione avvenuta un rootkit per garantirsi un accesso nelle prossime intrusioni e coprire le proprie tracce.

Nel server utilizzato come esempio per la relazione ho appunto evitato di installare il compilatore C, infatti, anche il kernel che ho preparato e' stato compilato su un'altra macchina e successivamente ho copiato l'immagine ottenuta nella directory **/boot** di **paranoid** (l'hostname del server).

In ogni caso mi e' parso necessario fare le precedenti considerazioni, facendo presente che esiste anche questa possibilita' per gestire i permessi associati a files e directory. Non necessitando io della flessibilita' offerta dalle **ACL** ho evitato di ricorrervi e quindi nel pieno rispetto della regola d'oro ho evitato anche di includere l'opzione **Ext3 POSIX Access Control Lists** in fase di ricompilazione del kernel.

Se invece si desidera farvi ricorso, oltre ad abilitare la precedente opzione nel kernel, e' necessario installare il pacchetto **acl** (che comprende appunto le utility **getfacl**, **setfacl** e **chacl**) con il comando **apt-get install acl**, ed aggiungere **acl** tra le opzioni di mount in **/etc/fstab**

Proseguendo nella configurazione, sarebbe una buona idea configurare **umask** in modo che i permessi alla creazione di nuovi files siano i piu' restrittivi possibile. Ad esempio si potrebbe modificare il valore di **umask** dell'utente **root** da **0022** a **0077** in **/root/.bashrc**, in modo da impedire lettura, scrittura ed esecuzione da altri utenti dei files creati, ad eccezione di quelli dove i permessi sono stati modificati manualmente con **chmod**.

Un'altra operazione che potrebbe essere opportuno fare e' la ricerca dei file **world writable** e di quelli con i bit **SUID/SGID** attivati.

Nel primo caso la ricerca e' stata fatta con il comando **find / -perm -2 ! -type l -ls**, che permette appunto di cercare i file scrivibili da tutti, escludendo dalla ricerca i link simbolici. Poiche' directory e file scrivibili da tutti potrebbero costituire una debolezza del sistema, e' importante sapere quali sono questi file, ed il motivo per cui i loro permessi sono impostati in tale modo.

Prevedibilmente entro questo elenco figurano le directory **/tmp**, e **/var/tmp**, oltre a tutta una serie di devices presenti sotto **/dev**.

Maggiore curiosita' e' stata destata invece dal fatto che risultassero **world writable** diversi files sotto le directory **/var/spool/postfix/public** e **/var/spool/postfix/private**. Andando a fare un controllo piu' approfondito si e' potuto notare che tali file erano in realta' pipe e socket, e che era un requisito essenziale impostarli con tali permessi in modo che l'mta potesse "comunicare" con gli utenti locali. I files sotto **/var/spool/postfix/private**, inoltre, si trovano in una directory accessibile esclusivamente all'utente **postfix**.

Questa considerazione, e' stata fatta per sottolineare la necessita' di documentarsi a dovere riguardo ai risultati ritornati dalla ricerca in questione, prima di apportare qualche modifica affrettata ai permessi, che possa portare a disservizi od altri problemi.

Lo stesso discorso si applica alla ricerca dei files con il bit **SUID** attivo, tramite il comando **find / -type f -perm -04000**. Nel server appena installato il risultato del comando **find** e' stato il seguente:

```
/bin/login
/bin/mount
/bin/ping
/bin/su
/bin/umount
/sbin/unix_chkpwd
/usr/bin/at
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/sudo
/usr/lib/pt_chown
/usr/sbin/pppd
/usr/sbin/pppoe
```

Gia' da questa lista si e' notata la presenza del demone **pppd** e **pppoe**, installati dalla distribuzione, ma non necessari poiche' nel nostro caso il server si trova dietro ad un router che si occupa di gestire la connessione. Si e' quindi provveduto a rimuoverli con i comandi **apt-get remove pppd** e **apt-get remove pppoe**.

Per tutti gli altri comandi, bisogna considerare caso per caso, se si vuole che tutti gli utenti possano eseguirli:

- **login** : il programma viene invocato automaticamente al login su una shell del sistema, ma puo' essere anche richiamato da linea di comando dagli utenti. Ad esempio un utente che volesse diventare root potrebbe eseguire il comando **exec login root**, per poi dovere immettere la password relativa a tale utente. Personalmente ho ritenuto non necessario lasciare il bit **SUID** attivo su tale programma(**chmod -s /bin/login**) , poiche' se dovesse esistere la reale necessita' di diventare root senza riloggarci, e' possibile utilizzare anche il comando **su**. Tra i due comandi esistono alcune differenze, ad esempio facendo **logout** dopo avere fatto un altro login con **su**, si ritorna alla shell chiamante. Facendo **logout** in seguito all'uso di **login**, invece, ci si ritrova al prompt per la richiesta di username e password.

Rispetto a **su**, **login** dovrebbe pero' utilizzare meno memoria. Penso che questa considerazione potesse avere maggior senso in passato, ma con le attuali configurazioni, non credo la cosa possa essere un fattore determinante.

Infine, a differenza di **su**, **login** mostra il nuovo utente nella lista generata dal comando **who**, anziche' l'utente che ha invocato il comando **su**.(fonte **comp.security.unix**).

Ho avuto modo di verificare che togliendo il bit **SUID** a **/bin/login**, ed eseguendo il comando **exec login root** con un utente "normale", viene comunque richiesta la password, ma anche se essa viene correttamente inserita l'autenticazione non va a buon fine, riportando l'utente al prompt di login.

- **chsh** : viene utilizzato per cambiare la shell di login di un utente. Naturalmente un utente normale ha la possibilita' di cambiare esclusivamente la propria shell. Personalmente non vedo la necessita' di consentire ad ogni utente il permesso di utilizzare una diversa shell da quella definita in **/etc/passwd**. Togliendo il bit **SUID** a **/usr/bin/chsh** viene comunque presentata all'utente la richiesta della propria password, ma anche in caso essa sia inserita correttamente il sistema risponde che tale password e' errata e riporta l'utente alla sua shell.
- **chfn**: consente di modificare le proprie finger information memorizzate nel file **/etc/passwd** . Ho scelto di rimuovere il bit **SUID** a **/usr/bin/chfn** poiche' non vedo la necessita' per un'utente di modificare i propri dati. Personalmente ritengo sarebbe meglio non registrare alcuna informazione oltre ad userid e password, poiche' esse potrebbero fornire indicazioni utili a qualche malintenzionato, quali il numero di telefono di un utente.

Tali informazioni possono essere visualizzate tramite il comando **finger**, o direttamente leggendo il file **/etc/passwd**, e potrebbero, ad esempio, essere sfruttate per cercare di carpire la password dell'utente o svolgere azioni di social engineering.

Tale programma consente inoltre di chiedere informazioni su un utente senza conoscere il nome del suo account. Esso infatti proverà a far corrispondere il nome reale con quello che voi gli date. Ad esempio, digitando **finger andrea**, mi verrà detto che per Andrea Beretta esiste un account andrea:

```
Login: andrea                      Name: Beretta Andrea
Directory: /home/andrea           Shell: /bin/bash
Office phone: 0363*****          Home phone: 0363*****
On since Fri Oct 28 08:25 (CEST) on tty1      2 hours 55 minutes idle
(messages off)
On since Fri Oct 28 09:42 (CEST) on tty0 from :0.0
No mail.
No Plan.
```

Sottolineo come, per default, sulla macchina debian usata come esempio, non sia presente il programma **finger**. L'ho installato io solo a titolo esemplificativo, ed ho provveduto a rimuoverlo nell'arco di pochi minuti. Allo stesso modo, le informazioni relative all'utente **andrea** sono state inserite solo in via temporanea e rimosse in seguito.

- **newgrp** : tale comando permette di cambiare il gruppo a cui appartiene un utente, modificando esclusivamente il proprio **GID**. Non volendo consentire il cambio di gruppo agli utenti diversi da root e' stato disabilitato il bit **SUID** per **/usr/bin/newgrp**.
- **gpasswd**: tale comando viene usato per la gestione del file **/etc/group**, relativo appunto ai gruppi. Poiche' ritengo piu' indicato che i gruppi possano essere gestiti esclusivamente da utenti specifici ho rimosso il bit **SUID** con **chmod -s /usr/bin/gpasswd**
- **passwd**: per questo comando ho deciso di lasciare attivo il bit **SUID**, poiche' mi sembra piuttosto sensata la possibilita' di consentire ad un utente di cambiare la propria password senza dovere ogni volta scomodare l'amministratore di sistema. Naturalmente saranno prese misure opportune per fare in modo che vengano impostate password sufficientemente complesse, come si vedra' piu' avanti parlando dell'autenticazione e di PAM.
- **at**: serve per fare eseguire job ad orari stabiliti. Senza il **SUID** bit attivo gli utenti non hanno la possibilita' di eseguirlo. Poiche' preferirei evitare di dare a tutti gli utenti la possibilita' di schedulare operazioni tramite **at**, ho lasciato comunque abilitato il bit **SUID**, e come indicato nella manpage del programma ho creato il file **/etc/at.allow**, in cui elencare gli utenti che possono utilizzarlo. In questo modo non viene considerato il file **/etc/at.deny**, e vengono abilitati all'utilizzo di **at** esclusivamente gli utenti definiti in **at.allow**.
Inoltre ho verificato che sebbene il **SUID** bit sia attivo, se un normale utente prova a programmare dei comandi per cui non abbia i privilegi necessari (nella fattispecie creare un file nella directory **/root** e ed eseguire il comando **/sbin/reboot**), essi non vengono comunque eseguiti.
- **sudo**: avendo la necessita' di fare eseguire alcuni comandi con i permessi di un'altro utente, si e' lasciato attivo il bit **SUID**.
- **su**: permette a un utente di diventare temporaneamente un altro, aprendo una shell con **UID** e **GID** opportuni. Se non viene indicato un utente, **su** sottintende root. Per potere svolgere il suo compito, il programma **su** richiede che il bit **SUID** a sia attivato. In questo modo si consente all'utente che l'ha invocato di ottenere i privilegi di root o dell'utente indicato.

Normalmente **su** dovrebbe venire usato dall'utente root, o da un processo che gia' abbia i privilegi dell'utente root, per diventare temporaneamente un altro utente. In casi del genere il processo che chiama **su** avra' gia' i

privilegi necessari e non c'è alcuna necessità di avere il **SUID** bit attivo.

La scelta di disattivare il bit **SUID** per **/bin/su** non va però presa alla leggera, perché possono esserci diversi casi in cui esso potrebbe essere indispensabile. Si immagini ad esempio il servizio **ssh** configurato in modo da non consentire l'accesso diretto all'utente **root**.

Nel mio caso ho scelto di lasciare il **SUID** bit attivo in modo da usarlo come alternativa al consentire il login diretto dell'utente **root** al sistema. Ho però provveduto a configurare opportunamente il tutto, in modo per potere utilizzare in maggior sicurezza il comando, come verrà illustrato durante la trattazione di **PAM**.

- **unix_chkpwd**: il programma in questione dovrebbe essere usato per il controllo delle password, ed è necessario lasciare attivato il **SUID** bit per consentirgli la lettura da **/etc/shadow**.
- **pt_chown**: consente di definire proprietario, gruppo e permessi di accesso ad uno pseudo terminale slave. Disattivare il bit **SUID** per **/usr/lib/pt_chown** potrebbe causare problemi di sicurezza anziché risolverli. Si immagini un processo lanciato da un utente diverso da **root** che necessita di ottenere accesso esclusivo ad un file od un device, ad esempio un emulatore di terminale, per cui sia richiesto che nessun altro utente possa leggere o scrivere dallo pseudo terminale associato. Lasciare aperto anche ad altri lo pseudo terminale associato all'emulatore potrebbe quindi non essere una buona idea.
- **ping, mount** ed **umount**: non ci si sofferma sul significato di questi comandi, poiché già trattati, ma si sottolinea come per ognuno di essi sia stato disattivato il bit **SUID**. Poiché essi potrebbero essere necessari per specifici utenti, si può comunque renderli utilizzabili tramite il comando **sudo**.

Sempre riguardo ai comandi **mount** ed **umount** si ricorda che già era stato considerato il fatto che un semplice utente non avrebbe dovuto avere la possibilità di usarli, visto che in **/etc/fstab** non è prevista neppure la possibilità di montare unità rimovibili quali floppy e cdrom. Infatti, nelle righe relative a tali dispositivi non è riportata l'opzione **user**.

Inoltre, presumendo che la macchina si trovi in un ambiente protetto, all'interno di un armadio rack, la cosa sarebbe poco sensata dato che l'utente potrebbe avere seri problemi nell'utilizzo "fisico" di un floppy disk o di un pen drive.

In conclusione si è optato per lasciare il bit **SUID** attivato solo per i comandi **passwd, su, sudo, unix_chkpwd, e pt_chown**.

Prima di passare alla configurazione opportuna del file **/etc/sudoers**, andrebbe effettuata anche la ricerca dei file con bit **SGID** attivo (**find / -type f -perm -02000**), che nel mio caso ha riportato i seguenti risultati :

```
/usr/bin/bsd-write
/usr/bin/chage
/usr/bin/crontab
/usr/bin/dotlockfile
/usr/bin/expiry
/usr/bin/wall
/usr/sbin/postdrop
/usr/sbin/postqueue
```

Come per i programmi con bit **SUID** attivo, anche questi debbono essere considerati singolarmente in modo da potere prendere la decisione più appropriata:

- **wall**: il programma è utilizzato per inviare un messaggio a tutti i terminali attivi. Ad esempio potrebbe essere utilizzato dall'utente **root** per avvisare di un reboot imminente della macchina o del fermo di alcuni servizi. Per potere scrivere sugli altri dispositivi, secondo le convenzioni, **wall** dovrebbe avere i privilegi del gruppo **tty** (come si può infatti notare da un **ls -l /usr/bin/wall**). Per cui, esso viene comunemente installato con il bit

SGID attivato, appartenendo al gruppo **tty**.

Secondo discussioni su **comp.security.unix** il programma non dovrebbe costituire un grosso problema di sicurezza, perche' viene comunque eseguito con i permessi del gruppo **tty** e non **root**. Inizialmente ho eliminato i permessi di esecuzione per **other** (**chmod o-x /usr/bin/wall**), temendo che un utente malintenzionato potesse creare uno script che tramite tale comando potesse "floodare" i terminali di altri utenti, rendendo di fatto impossibile il loro lavoro.

Successivamente li ho pero' riabilitati, poiche' ho avuto modo di verificare che il gruppo **tty** non ha permessi di scrittura sui vari **/dev/ttyN** aperti. L'esecuzione di **wall** da parte di un utente normale mostrerebbe quindi i messaggi esclusivamente sul proprio terminale e non su quelli altrui.

- **chage**: consente di visualizzare e modificare le informazioni relative alla validita' della password di un utente. Il proprietario del file e' l'utente **root**, mentre il gruppo e' **shadow**. In un sistema che utilizza le **shadow** password, e' necessario lasciare lo **SGID** bit impostato perche' un utente possa vedere le informazioni relative alla validita' della propria password (**chage -l**). In caso contrario **chage** non sarebbe in grado di accedere in lettura al file **/etc/shadow**.

In ogni caso, anche lasciando il bit **SGID** attivo, un utente puo' solo visualizzare le proprie informazioni, ma non modificarle. Se ad esempio l'utente **davide** per pigrizia volesse impostare la scadenza della sua password tra 99999 giorni con il comando **chage -M 99999 davide**, il sistema risponderebbe con **chage: permission denied**.

- **expiry**: secondo la manpage del programma, dovrebbe servire per verificare la validita' della password e forzarne il cambio quando necessario. L'utente proprietario del file **/usr/bin/expiry** e' **root**, mentre il gruppo e' **shadow**. Poiche' il programma e' eseguibile anche da un normale utente, ho lasciato attivo il bit **SGID**, supponendo esso sia necessario per consentire l'accesso al file **/etc/shadow**.
- **crontab**: il proprietario del file e' **root**, mentre il gruppo **crontab**. Tale programma serve per modificare il **crontab** degli utenti. L'utente **root** ha la possibilita' di creare **crontab** per tutti gli utenti, mentre un utente normale solo per se' stesso. Tali file vengono poi letti dal demone **cron** per eseguire con le scadenze fissate i comandi in essi definiti.

Questi files sono memorizzati nella directory **/var/spool/crontab**, sulla quale hanno i permessi di scrittura solamente l'utente **root** ed il gruppo **crontab**. E' quindi necessario lasciare il bit **SGID** attivo per consentire la scrittura e la modifica di tali files. Non volendo pero' consentire a tutti gli utenti la possibilita' di creare un proprio **crontab** per schedare delle operazioni, si e' creato il file **/etc/cron.allow**. In questo modo solo gli utenti definiti all'interno di tale file potranno usare **crontab**.

- **dotlockfile**: e' un utility per la creazione, test e rimozione di lockfile, ad esempio per bloccare o sbloccare delle mailbox. Essendo la directory **/var/mail** scrivibile solo dall'utente **root** e dal gruppo **mail**, per potervi creare dei lockfile e' necessario lasciare il bit **SGID** attivo su **/usr/bin/dotlockfile**. In alternativa si potrebbe rimuovere il bit **SGID** e rendere world writable la directory **/var/mail**, ma non mi sembra una soluzione migliore della precedente: in questo modo oltre a creare un file di lock tramite **dotlockfile**, diventerebbe possibile anche creare altri file entro tale directory.
- **bsd-write**: la manpage del programma riporta alla man page di **write**, ed il comando **write** e' in realta' un link simbolico a **/usr/bin/bsd-write**. Non avendo trovato documentazione specifica per il comando **bsd-write**, mi sono dunque rifatto a quella di **write**. Il discorso relativo ai permessi e' pressoché identico a quello fatto riguardo a **wall**, la differenza tra i due programmi sta nel fatto che **bsd-write** consente di selezionare il terminale o l'utente a cui inviare il messaggio.

Ho lasciato il bit **SGID** attivo, poiche' il gruppo associato a **/usr/bin/bsd-write** e' **tty**, ed esso non ha permessi di scrittura sulle diverse **/dev/ttyN**. Non c'e' quindi il problema di un eventuale flooding da parte di un utente diverso da **root**.

- **postdrop**, **postqueue**: entrambi i programmi hanno come user **root** e **postdrop** come gruppo, e fanno parte dell'mta **postfix**. Per default **debian** installa **exim** come mta per gestire la posta locale. Trovandomi piu' a mio

agio con **postfix**, ed avendo esso una migliore reputazione in fatto di sicurezza, l'ho sostituito con quest'ultimo. Hp lasciato attivo il bit **SGID** per questi due programmi poiche' necessario per consentire loro di comunicare con i processi appartenenti ai demoni interni al sistema di posta (fonte <http://www.openskills.info>). Questo sistema di posta, infatti, anziche' essere costituito da un unico grosso programma che effettua la maggior parte delle azioni e' suddiviso in diversi programmi che svolgono una azione specifica. Esiste poi un programma principale che si occupa di chiamare il programma giusto quando necessario e di gestire le varie azioni. Poiche' il progetto non tratta la sicurezza locale e non la configurazione di un mailserver, non ci si concentra in dettaglio sulle caratteristiche ed il setup di **postfix**.

Definiti quali debbano essere i programmi con i bit **SUID** e **SGID** attivi, e' necessario iniziare a considerare delle eccezioni nell'esecuzione di alcuni programmi, consentendone l'utilizzo anche a specifici utenti oltre che root. Per fare cio' mi sono avvalso del programma **sudo**.

Esso permette, infatti, di eseguire programmi con i permessi di un altro utente (e quindi anche di root), come per il bit **SUID** che consente invece di eseguire un programma con i privilegi del proprietario. **Sudo** consente pero' una maggiore flessibilita', ad esempio definendo chi siano gli utenti abilitati ad eseguire tali comandi, tramite il file di configurazione **/etc/sudoers**.

A titolo di esempio, si e' ipotizzato che oltre all'utente **root**, esistano un gruppo di amministratori con pieni poteri sulla macchina, tra cui rientra l'utente **andrea** ed un gruppo di operatori con privilegi intermedi, tra cui l'utente **davide**.

Avendo precedentemente installato **sudo** con il comando **apt-get install sudo**, e' stato quindi editato il file **/etc/sudoers**. Anziche' utilizzare il classico **vi**, e' pero' stato utilizzato il comando **visudo**, poiche' esso consente di editare **/etc/sudoers** in maniera sicura, lockando il file ed impedendo scritture simultanee. Esso inoltre esegue anche una serie di controlli, per verificare che quanto inserito rispetti la sintassi prevista.

Il file **/etc/sudoers** preparato e' quindi stato il seguente:

```
#Defaults
Defaults logfile=/var/log/sudo.log, log_year
#User alias specification
User_Alias OPERATORS = davide
#Command alias specification
Cmnd_Alias BACKUP = /sbin/backup_tar.sh
Cmnd_Alias KILL = /bin/kill
Cmnd_Alias PERMISSION = /bin/chmod, /usr/bin/setfacl
Cmnd_Alias PRINT = /usr/bin/cancel, /usr/bin/lprm,
                  /usr/sbin/lpc, /etc/init.d/cupsys
Cmnd_Alias SHUTDOWN = /sbin/halt, /sbin/reboot,
                  /sbin/shutdown
Cmnd_Alias SERVICES = /etc/init.d/
Cmnd_Alias USERMGMT = /usr/sbin/adduser, /usr/sbin/addgroup,
                  /usr/bin/chage, /usr/bin/expiry,
                  /usr/bin/passwd, /usr/bin/chsh,
                  /usr/bin/chfn, /usr/bin/gpasswd,
                  /usr/sbin/deluser
Cmnd_Alias MOUNT = /bin/mount, /bin/umount
#User privilege specification
root ALL = (ALL) ALL
OPERATORS paranoid = BACKUP, KILL, PERMISSION, PRINT, SHUTDOWN,
```

SERVICES, USERMGMT

Nel file oltre all'utente **root** e' stato previsto un alias per gli utenti, **OPERATORS**, pensato per eseguire piccoli interventi sulla macchina senza richiedere l'intervento dell'amministratore di sistema. Una soluzione del genere potrebbe essere adatta per un ambiente dove oltre all'amministratore principale, presumibilmente operato di lavoro, vi siano anche altri utenti abbastanza abili e in cui si abbia fiducia sufficiente da consentirgli l'esecuzione di alcune operazioni particolari. Non e' invece presente alcuna voce per l'utente **andrea** o per un gruppo di utenti con permessi di amministratori, poiche' ad essi viene consentita la possibilita' di impersonare l'utente **root** tramite il comando **su**, come verra' illustrato successivamente, trattando l'argomento **PAM**.

Per migliorare la leggibilita' e la manutenibilita' del file, anziche' stilare solamente una lunga lista di comandi per tale alias, sono stati definiti degli ulteriori alias con dei gruppi di comandi. Per la precisione tali alias sono:

- **BACKUP**: comprende esclusivamente lo shellscript **backup_tar.sh**, che sara' illustrato successivamente, trattando il backup del sistema.
- **KILL**: comprende solo il comando **kill**, usato per inviare segnali ai processi. Si ipotizzi il caso di un utente che abbia schedulato un programma da lui scritto e compilato (ipotizzando egli possa utilizzare un compilatore), che a causa di un errore inizia a forkare e creare un numero esagerato di processi figli, o piu' semplicemente causi un consumo di risorse troppo elevato. Se ne tale utente ne l'amministratore di sistema fossero reperibili, potrebbe essere un problema uccidere il processo che e' all'origine del problema. Consentire l'utilizzo del comando **kill** ad utenti selezionati potrebbe rendere meno probabile un simile scenario.
- **PERMISSION**: comprende i comandi necessari ad impostare i permessi su file e cartelle. Si impedisce che ciascun utente possa cambiare i permessi ai propri file, facendo in modo che essi abbiano i privilegi definiti tramite **umask**.

In caso di necessita' i permessi potranno essere cambiati da uno degli amministratori o uno degli utenti elencati in **OPERATORS**, oppure sara' sufficiente aggiungere al file **/etc/sudoers** la riga `nome_utente paranoid = PERMISSION`.

- **PRINT**: comprende una serie di comandi relativi alla gestione della stampante e della coda di stampa. In questo modo gli utenti elencati in **OPERATORS** potranno gestire la stampante e risolvere eventuali semplici problemi ad essa relativi, senza l'intervento dell'amministratore di sistema, come ad esempio la cancellazione di alcuni lavori dalla coda di stampa. Sulla macchina usata per il progetto, in realta' non e' stato definito questo alias, poiche' ad essa non e' collegata alcuna stampante, ma e' stata citato a scopo dimostrativo.
- **SERVICES**: comprende tutti gli script di init, compresi nella directory **/etc/init.d**. Potrebbe essere utile per consentire agli utenti elencati in **OPERATORS** di fermare/avviare/riavviare i diversi servizi. In caso si desideri una maggiore granularita' e' possibile definire i singoli servizi, magari sotto alias diversi. Questa soluzione, ad esempio, potrebbe essere utile nel caso si presentasse la necessita' di uccidere il demone relativo a qualche servizio e riavviarlo subito dopo.

Sempre riguardo agli script della directory **/etc/init.d**, potrebbe essere una buona idea impostarli in modo che solo l'utente **root** possa vederne il contenuto ed eseguirli, con il comando **chmod -R 700 /etc/init.d/**.

- **SHUTDOWN**: i comandi necessari per il riavvio e lo spegnimento della macchina. Si immagini una situazione di emergenza in cui sia richiesto lo spegnimento della macchina, in assenza dell'amministratore di sistema. Staccare brutalmente l'alimentazione potrebbe non essere la soluzione ideale. In questo modo un utente facente parte di **OPERATORS** sarebbe abilitato a farlo.
- **USERMGMT**: comandi necessari per la creazione, cancellazione e modifica di gruppi e utenti. In questo modo l'amministratore di sistema puo' essere sollevato dal compito della creazione e gestione degli account.

All'esecuzione dei comandi tramite **sudo**, verra' chiesta una password, non quella di **root**, come nel caso di **su**, ma quella dell'utente che ha eseguito il comando. Volendo e' possibile eliminare questa richiesta, tramite la direttiva

NOPASSWD nel file `/etc/sudoers`, ma ho preferito lasciarla, per evitare che qualche utente possa eseguire dei comandi con **sudo** se un operatore dovesse lasciare la sua postazione incustodita.

Avendo l'assoluta certezza che nessuno di non autorizzato possa accedere alla tastiera si potrebbe anche disattivare tale opzione. Nel caso invece venga anche consentito l'accesso al sistema via **ssh**, ad esempio, ritengo sarebbe particolarmente indicato mantenerla, poiche' non e' detto che in caso di una connessione da remoto, nel luogo da cui viene eseguito l'intervento vi sia un controllo "fisico" degli accessi adeguato.

Per evitare la richiesta della password ad ogni comando, si potrebbe impostare l'opzione **passwd_timeout**, che consente di definire il numero di minuti che devono trascorrere perche' **sudo** chieda nuovamente l'inserimento della password.

Un ulteriore caratteristica interessante, legata a **sudo**, e' anche il fatto che esso provveda ad inviare all'amministratore di sistema una mail per avvisare ogni caso in cui un utente tenti di eseguire tramite **sudo** un comando per il quale non era autorizzato.

Teoricamente sarebbe anche possibile andare a guardarsi il file di log di sudo (nel caso riportato `/var/log/sudo.log`), ma visti tutti i problemi di privacy legati al dlgs 196/2003, la possibilita' di essere avvisati automaticamente in caso di violazione e' di estrema utilita'.

Se ad esempio l'utente davide, cercasse di editare il file di configurazione di **lilo**, tramite il comando **sudo vi /etc/lilo.conf**, l'amministratore riceverebbe una mail dal subject `*** SECURITY information for paranoid.localdomain ***` contenente il seguente testo :

```
paranoid.localdomain : Oct 31 11:31:23 : davide : user NOT in sudoers ;
TTY=tty2 ; PWD=/home/davide ; USER=root ; COMMAND=/usr/bin/vi /
etc/lilo.conf
```

Infine, un ultima considerazione che si potrebbe fare relativamente alle misure di sicurezza relative al filesystem, riguarda l'adozione di un filesystem **RAID**, sia esso gestito via software (tramite il kernel) oppure via hardware con un controller.

Non avendo limiti di budget, la scelta migliore dovrebbe essere l'adozione di una soluzione hardware, per motivi di performance e per non appesantire il kernel del sistema operativo di un ulteriore compito.

Esistono infatti diverse offerte, in ogni fascia di prezzo, partendo da poche decine di euro a svariate centinaia. Se il budget e' in grado di consentire l'acquisto di hardware di buona qualita' e' sicuramente la strada da preferire. In caso contrario, anziche' acquistare schede di affidabilita' dubbia, tanto vale optare per una soluzione software, come appunto la gestione del **RAID** tramite il kernel.

Bisogna inoltre considerare che il Kernel non consente di creare sistemi **RAID** "nidificati" come ad esempio **RAID 0+1** o **RAID 1+0**, per i quali bisognera' ricorrere necessariamente ad una soluzione hardware.

Poiche' esistono diversi livelli di **RAID**, quale potrebbe essere il piu' adatto allo scopo? Ritengo sia da scartare l'utilizzo del solo **RAID-0** o **striping**, poiche' esso si limiterebbe a ripartire i dati su due o piu' dischi senza alcuna informazione di parita'. Esso potrebbe essere utilizzato per aumentare le prestazioni del sistema, poiche' consentirebbe, in condizioni ottimali, letture e scritture piu' veloci.

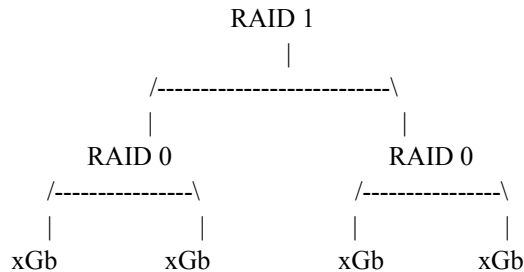
L'affidabilita' sarebbe pero' inversamente proporzionale al numero di dischi, essendo equivalente all'affidabilita' media di un singolo disco divisa per il numero dei dischi. Poiche' i dati sono divisi tra i diversi dischi, e' sufficiente che si guasti uno di essi perche' i dati in esso contenuto non siano piu' disponibili.

Piu' adatto ai fini di garantire una maggiore sicurezza dei dati, e' invece il sistema **RAID-1** o **mirroring**, in cui i dati sono replicati su tutti i dischi dell'array. In questo modo le letture possono avvenire piu' velocemente (leggendo i frammenti contemporaneamente da ogni disco), mentre la scrittura avra' la stessa velocita' di un singolo disco. In questo modo, pero', in caso di rottura di un disco, i dati saranno ancora disponibili sugli altri elementi dell'array.

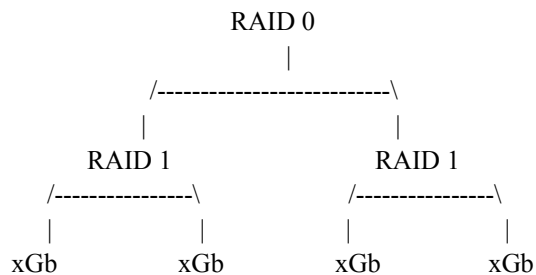
Volendo, un'altra possibile soluzione avrebbe potuto essere il **RAID-5**, che avrebbe richiesto meno unita di un **RAID-1**, ma sarebbe risultato meno robusto, potendo sostenere la perdita di un solo disco per gruppo.

Un'ulteriore possibilita' e' quella di combinare tra loro i livelli, ad esempio nel sistema **RAID 0+1**. In tale sistema, i dati vengono sia replicati che condivisi tra diversi dischi:

Ad esempio, si ipotizza di avere uno spazio disponibile di $2 \times x$ Gb, diviso tra due sistemi. Il vantaggio rispetto ad un sistema basato esclusivamente su **RAID-0** e' che in caso di rottura di uno dei dischi in **RAID-0** e' ancora possibile accedere ai dati in essi contenuti tramite l'altro **RAID-0**. Tale sistema non puo' pero' sopportare la rottura dei due dischi nei due **RAID-0**.



Una soluzione piu' robusta, pero', dovrebbe essere invece il sistema **RAID 1+0**, sostanzialmente simile al precedente, ma con i livelli invertiti:



Nell'esempio riportato si hanno due insiemi di dischi in **RAID-1**, ognuno con x Gb di spazio, uniti in un sistema **RAID-0** per uno spazio di $2 \times x$ Gb.

Con questa soluzione, ogni disco dei sistemi **RAID-1** puo' danneggiarsi senza perdita di dati, che saranno comunque presenti sugli altri dischi del rispettivo **RAID-1**. Se pero' dovesse danneggiarsi anche il rimanente disco del sistema **RAID-1**, tutte le informazioni dell'intero sistema andrebbero perse, poiche' il livello **RAID-0** avrebbe a disposizione solo le parti di dati presenti sul secondo **RAID-1**.

Gli esempi sono stati fatti con due dischi, per motivi di semplicita', ma si potrebbero utilizzare anche piu' dischi, a seconda del budget disponibile.

Integrita' del filesystem

Restando sull'argomento filesystem, sarebbe di estrema importanza avere la possibilita' di verificare l'integrita' di alcuni files, in modo da capire se essi sono stati modificati. Ad esempio, in seguito ad una compromissione, l'attacker potrebbe avere installato un rootkit non lkm che abbia provveduto a sostituire alcuni programmi di sistema quali **grep**, **ls**, **ps**, **netstat** ecc., in modo da nascondere appunto la presenza del rootkit stesso.

Una possibile soluzione potrebbe essere calcolare gli hash dei files che si desidera monitorare, in un momento in cui si abbia la sicurezza che la macchina non sia stata compromessa (ad esempio prima di connetterla in rete). In seguito ricalcolando gli hash di tali file e confrontandoli con i valori precedenti e' possibile verificare se essi abbiano subito delle modifiche o sostituzioni. Infatti, anche la piu' piccola modifica di un file causa la modifica del proprio hash. E' inoltre quasi impossibile creare due files diversi con lo stesso hash.

Naturalmente nel calcolo degli hash e' necessario utilizzare un algoritmo che non sia stato violato. Per tale motivo vanno scartati algoritmi quali **MD2** o **MD4**, in favore di **MD5** o **SHA1**.

Calcolare e verificare l'hash manualmente per ogni file sarebbe un lavoro estremamente improduttivo, che dovrebbe essere svolto troppo frequentemente. E' infatti necessario ricalcolarsi l'hash dei files ad ogni update, ed aggiornare il proprio database usato per il confronto, poiche' esso non rispecchierebbe piu' lo stato attuale del sistema.

Una possibile soluzione potrebbe essere la creazione di una serie di shellscript da eseguire con cadenza regolare, che calcolino gli hash di determinati files (ad esempio i programmi in **/bin**, **/usr/bin**, **/sbin** e **/usr/sbin**), e li confrontino tramite il comando **diff** con i valori "originali". Differenze tra tali valori indicherebbero una modifica o sostituzione del file.

Questa soluzione sebbene abbastanza rudimentale, richiederebbe un certo impegno da parte dell'amministratore per creare una serie di script ad hoc. Fortunatamente esistono software pensati appositamente per questo scopo, estremamente flessibili nella definizione dei controlli e con report decisamente piu' dettagliati.

Da alcune ricerche effettuate con google i file integrity checker piu' ricorrenti in ambiente linux sono **Tripwire**, **AFICK** ed **AIDE**.

Ho scartato il primo perche' di natura commerciale, mentre la sua versione free non viene aggiornata da Marzo 2001. La scelta e' quindi stata ristretta tra **AFICK** ed **AIDE**: entrambi supportano il calcolo sia degli hash **MD5** e **SHA1**.

La scelta e' infine caduta su **AIDE** perche' gia' presente nei repository di debian sarge. In questo modo posso e' possibile usufruire degli eventuali updates tramite **apt**, senza dovere tenere controllato il sito web del progetto e scaricare ogni volta le nuove versione.

L'installazione al solito e' stata fatta tramite **apt** con **apt-get install aide**, in modo da risolvere automaticamente le eventuali dipendenze (come ad esempio la libreria **libmhash**).

Completata l'installazione, ho editato il file di configurazione **/etc/aide/aide.conf**:

```
database=file:/var/lib/aide/aide.db
database_out=file:/var/lib/aide/aide.db.new
database_new=file:/var/lib/aide/aide.db.new
gzip_dbout=yes
report_url=stdout
report_url=file:/var/log/aide/aide.log
#binari e librerie
BINLIB = p+i+n+u+g+s+b+m+c+sha1+md5
#file di configurazione
CONF = i+n+u+g+s+b+m+c+sha1+md5
#log
LOGS = p+u+g
```

```

#risorse da monitorare
#Kernel
/boot                BINLIB
#binari
/bin                 BINLIB
/sbin                BINLIB
/usr/bin              BINLIB
/usr/sbin             BINLIB
/usr/local/bin        BINLIB
/usr/local/sbin       BINLIB
#librerie
/lib                  BINLIB
/usr/lib               BINLIB
/usr/local/lib        BINLIB
#files configurazione
/etc                  CONF
#logfiles
/var/log              LOGS

```

Nelle prime tre righe vengono definiti rispettivamente file da utilizzare come database nei confronti, il file generato da **aide** in fase di inizializzazione o aggiornamento, ed il file da confrontare con **aide.db** quando viene utilizzata l'opzione **--compare**.

All'interno del file sono state definite le tre seguenti regole per il monitoraggio :

- **BINLIB**: questa regola verifica se sono avvenuti cambi di permessi, del numero di inode, numero dei link simbolici verso il file, proprietario o gruppo associato, dimensione del file, numero dei blocchi, data di modifica del file e data di modifica dell'inode. Vengono inoltre valutati sia gli hash **md5** e **sha1** del file.

Con ogni probabilita' il calcolo di un singolo hash sarebbe stato piu' che sufficiente, vista la difficolta' di collisioni, ma si e' preferito essere paranoici. Calcolando due hash penso si possa avere la ragionevole certezza che un file che abbia mantenuti invariati tali valori non sia stato modificato: e' praticamente impossibile che due sequenze di bytes differenti possano generare la stessa coppia di hash **md5** e **sha1**.

Tale regola viene utilizzata per la verifica dell'integrita' dei programmi contenuti nelle directory **/bin**, **/sbin**, **/usr/sbin**, **/usr/bin**, **/usr/local/bin**, **/usr/local/sbin** e delle diverse librerie in **/lib**, **/usr/lib** e **/usr/local/lib**. Oltre che per queste directory, vi si fa ricorso anche per **/boot**, in modo da potere verificare eventuali modifiche o manomissioni all'immagine del kernel o della **System.map** (NOTA: su sistemi con kernel modulari, ad ogni riavvio del sistema aide segnalera' una variazione delle date di creazione e modifica di una serie di file quali **modules.dep**, **modules.portmap**, **modules.pcimap** ecc. poiche' essi vengono riscritti ad ogni riavvio del sistema. Usando sul server un kernel monolitico, non si pone comunque il problema).

- **CONF**: esegue gli stessi controlli regola **BINLIB**, ad eccezione della modifica dei permessi. Tale scelta e' stata dovuta all'impressionante numero di falsi positivi nella directory **/etc** ottenuti con tale regola. Essa viene infatti utilizzata per la verifica dei files di configurazione posti sotto la directory **/etc**.

Inoltre, alcuni files come **/etc/motd**, **/etc/mtab** (l'elenco delle partizioni montate), **/etc/adjtime** (file di configurazione per l'aggiustamento dell'orologio hardware) ed **/etc/network/ifstate** (contiene lo stato delle interfacce di rete) vengono scritti ad ogni avvio del sistema. Quindi in caso di reboot della macchina e' necessario tenere presente questo fatto, poiche' aide segnalera' la loro modifica.

Volendo sarebbe possibile escludere tali file dal monitoraggio, ma poiche' la macchina utilizzata per i test svolga la funzione di server, si presume che non debba venire spenta o riavviata con particolare frequenza, e quindi si e'

preferito monitorare anch'essi.

- **LOGS:** verifica solo i cambi di permessi, gruppo e proprietario. Utilizzando tale direttiva per i log di sistema, si e' evitato di controllare anche variazioni di dimensione, hash e data dell'ultima modifica. In caso contrario, l'amministratore finirebbe con il ricevere continuamente degli alert relativi alle modifiche dei files di log, che vengono comunque effettuate regolarmente ed in maniera del tutto legittima.

Creato il file di configurazione e' stato inizializzato il database da usare come "immagine iniziale" del sistema con il comando **aide --init**. Con questo comando e' stato creato il file **/var/lib/aide/aide.db.new**, contenente l'hash di tutti i file indicati tra le risorse da monitorare, oltre a diverse altre informazioni quali proprietari, permessi, date di modifica ecc, a seconda delle regole specificate.

Poiche' nel file di configurazione ho indicato **aide.db** come database e' stato necessario anche un **cp /var/lib/aide/aide.db.new /var/lib/aide/aide.db**, visto che il database generato si chiamava **aide.db.new**.

Per verificare l'integrita' dei files monitorati e' sufficiente importire il comando **aide --check**. Esso confronta lo stato attuale del sistema con quanto memorizzato in **aide.db**. In caso di discrepanze vengono mostrate a video le differenze rilevate, come cancellazione o creazione di nuovi files, modifiche dei permessi, cambio di valori dei checksum e cosi' via.

Una funzionalita' simile e' quella dell'opzione **--compare**, che esegue pero' un confronto tra i due database **aide.db** e **aide.db.new**.

Perche' il sistema di controllo dell'integrita' possa funzionare correttamente, senza fare impazzire l'amministratore di sistema con una marea di falsi allarmi, e' fondamentale ricordarsi di aggiornare il database **aide.db** ad ogni installazione/update di qualche software, od alla modifica di qualche file di configurazione. In caso contrario, sebbene le modifiche effettuate fossero legittime, **aide** rilevarebbe delle differenze rispetto alla precedente immagine del sistema, segnalandole in modo opportuno. Per l'aggiornamento del database viene utilizzata l'opzione **update**: **aide --update**.

Anche in questo caso **aide** generera' un file chiamato **aide.db.new**, che dovra' essere copiato al posto del precedente **aide.db**.

Personalmente, farei precedere l'installazione aggiornamento dei pacchetti, da un ulteriore check con **aide**, in modo da verificare che non ci siano state eventuali modifiche dall'ultimo controllo.

Una volta generata l'immagine iniziale del sistema e schedulata l'esecuzione di **aide** il problema dei root non lkm o di manomissioni in seguito ad intrusioni potrebbe sembrare risolto, ma in realta' esistono ancora punti deboli piuttosto evidenti.

Essendo il database utilizzato per i confronti memorizzato sulla macchina, nulla vieterebbe ad un attacker che abbia fatto un'escalation dei privilegi fino al livello di root, di installare un rootkit, generare una nuova immagine dei file del sistema e sostituirla al vecchio database.

Una soluzione non troppo furba potrebbe essere quella di montare tale database sotto una partizione in sola lettura, ma un'attacker minimamente scaltro una volta diventato root potrebbe rimontarla con anche permessi in scrittura e sostituire il database.

Una possibile alternativa, potrebbe essere la copia del database su un CD. In questo modo il confronto avverrebbe con l'immagine registrata sul cd, che non potrebbe venire manomessa dall'attacker. Questo sistema non sarebbe pero' il massimo della comodita', poiche' comporterebbe la necessita' di masterizzare il file usato come database in seguito ad ogni aggiornamento o installazione di nuovo software. Personalmente ritengo che possa essere adatta finche' si debba monitorare una sola macchina, o comunque un numero ristretto. In caso contrario la procedura sarebbe un po' troppo macchinosa.

Un'altra alternativa, potrebbe essere la copia del database tramite un canale cifrato (**ssh/scp**) su di un secondo sistema.

Pur dovendo monitorare una sola macchina, per scopo didattico, ho comunque adottato questa soluzione.

Per prima cosa ho installato il server **ssh** su **paranoid** (la macchina da monitorare) ed il client su un secondo host

(**stargazer**, la macchina su cui verranno copiati i database), come sempre con **apt** (**apt-get install ssh**).

Avendo la necessita' di effettuare connessioni **ssh** e copie tramite **scp** all'interno di uno shellscrip, anziche' utilizzare l'autenticazione tramite password ho fatto ricorso a un meccanismo con chiave asimmetrica.

Per evitare di utilizzare l'utente **root**, inoltre, su entrambe le macchine ho creato un utente chiamato **rfc**, e sono ricorso ancora all'utilizzo di **sudo**.

Sull'host **stargazer** ho effettuato il login come utente **rfc**, ed ho creato una coppia di chiavi tramite il comando **ssh-keygen -t dsa**, memorizzandole nella directory **/home/rfc/.ssh**. Ovviamente durante la generazione delle chiavi non ho specificato alcuna password, altrimenti avrei sempre avuto il problema della richiesta delle password all'interno degli shellscrip.

Come si puo' notare, la chiave generata e' del tipo **dsa**. La scelta e' dovuta al fatto che il server ssh in esecuzione su **paranoid** e' configurato per utilizzare la versione **2** del protocollo, poiche' ritenuta piu' sicura rispetto alla **1**.

La chiave pubblica **id_dsa.pub** e' stata poi copiata su **paranoid** con il comando **scp /home/rfc/.ssh/id_dsa.pub rfc@paranoid:**. Naturalmente non avendo ancora inserito tale chiave tra quelle autorizzate per il server **paranoid**, e' stato necessario inserire la password dell'utente **rfc** alla richiesta.

Successivamente sull'host **paranoid**, e' stata inserita la chiave precedentemente copiata tra quelle autorizzate all'interno del file **/home/rfc/.ssh/authorized_keys2**, tramite il comando **cat id_dsa.pub >> /home/rfc/.ssh/authorized_keys2**.

In questo modo l'utente **rfc** e' in grado di connettersi ed eseguire comandi sull'host **paranoid** senza l'inserimento di alcuna password.

Sono consapevole che consentire la connessione dell'utente **rfc** senza la richiesta di una password rappresenti un compromesso a livello di sicurezza. E' comunque richiesta la sua chiave privata per potere effettuare l'accesso **ssh** senza la conoscenza della password. Tale chiave e la directory ove essa e' memorizzata hanno i privilegi minimi possibili, rispettivamente **600** e **700**. Esse sono quindi accessibili esclusivamente agli utenti **rfc** e **root**.

Per ottenere tale chiave e' quindi necessario riuscire ad ottenere accesso all'host **stargazer** (sul quale non e' in esecuzione il servizio **ssh**), che non e' comunque accessibile dall'esterno della rete.

In caso **stargazer** venga compromesso da un attacker, egli sara' in grado di accedere a **paranoid** come utente **rfc**, ma si ricorda che tale utente non ha alcun privilegio particolare oltre all'esecuzione di uno script **rfc_create_db.sh** tramite **sudo**, come verra' illustrato successivamente.

Sono consapevole che una volta ottenuta la chiave privata dell'utente, e avendo accesso come **rfc** all'host **paranoid**, sarebbe possibile cercare un escalation dei privilegi fino a livello di **root**, ma ritengo tutto cio' un compromesso accettabile.

Ai fini della sicurezza del sistema ritengo piu' probabile la manomissione del database con i checksum dei file se esso fosse memorizzato in locale. Dovendo scegliere tra le due alternative, ho preferito il male minore, ovvero consentire l'accesso tramite l'uso della crittografia asimmetrica all'utente **rfc**.

A scanso di equivoci, inoltre, sottolineo come il login sia possibile esclusivamente dall'host **stargazer** verso **paranoid**, e non viceversa, non essendo appunto in esecuzione il server **ssh** sulla prima macchina.

Oltre a permettere questo tipo di autenticazione, su **paranoid** e' stato modificato il file **/etc/ssh/sshd_config**, aggiungendo o modificando le seguenti opzioni :

```
Protocol 2
ListenAddress 192.168.1.4
PermitRootLogin no
PermitEmptyPasswords no
LogLevel INFO
SyslogFacility LOCAL0
AllowUsers rfc@192.168.1.2
```

Come si puo' vedere dallo stralcio del file di configurazione viene indicato il protocollo **2**, ed il server **ssh** e' in ascolto esclusivamente sull'indirizzo IP **192.168.1.4** . Nel mio caso il server ha una sola scheda di rete, quindi avrei anche potuto omettere tale direttiva. Per lan di maggiori dimensioni, al di fuori di un contesto casalingo, riterrei opportuno l'utilizzo di due schede di rete con due diversi indirizzi, mettendo in ascolto il server ssh solo su una di esse, facendo in modo che tale indirizzo sia accessibile solo della rete interna, magari sfruttando la topologia delle rete stessa.

E' inoltre stata cambiata anche la facility dei messaggi generati per il log server, in modo da poterli memorizzare in un file separato.

Tra le altre opzioni vengono negati l'accesso diretto via **ssh** dell'utente **root**, la possibilita' di usare password vuote (da non confondersi con le password che possono essere associate alle chiavi private degli utenti), ed impostata la verborita' dei log al livello **INFO**, in modo da avere maggiori informazioni rispetto al livello di default **ERROR**, ma senza arrivare a violare la privacy con il livello **DEBUG**.

Tramite l'ultima direttiva, ovvero **AllowUsers**, viene consentito l'accesso al solo utente **rfc** dall'host **stargazer**. Come si puo' notare, e' stato utilizzato l'indirizzo IP di **stargazer** anziche' il suo hostname, perche' sebbene fosse inserito in **/etc/hosts**, il demone **sshd** non riusciva a risolverlo.

Per cercare di garantirsi una maggiore sicurezza, consentendo l'accesso **ssh** esclusivamente dall'host **stargazer**, e' stato inoltre utilizzato **tcpwrapper (tcpd)** per lasciare passare solo connessioni da indirizzi IP ritenuti affidabili.

Tcpd, consente infatti di filtrare e monitorare le richieste verso un particolare servizio di rete. Il programma non ha nessuna interazione con il lato client ed e' completamente trasparente agli occhi degli utenti.

Lo stesso discorso vale per il lato server, una volta che il wrapper ha deciso di accettare la connessione, passa il controllo al server che non e' a conoscenza dell'esistenza di questo filtro. Il wrapper infatti e' attivo solo all'atto dell'instaurazione della connessione: una volta accettata o scartata, non interagisce piu' nella comunicazione.

Per utilizzare il wrapper ho fatto in modo che il server **ssh** venisse avviato tramite il superdemone **inetd**, aggiungendo la seguente riga in **/etc/inetd.conf** :

```
ssh    stream      tcp    nowait      root    /usr/sbin/tcpd
/usr/sbin/sshd    -i
```

Ssh indica il servizio, **stream** il tipo di socket, **tcp** il protocollo, **nowait** indica al server di non aspettare che venga rilasciato il socket per rimettersi in ascolto sulla relativa porta, **root** il nome dell'utente con cui gira **sshd**, **/usr/sbin/tcpd** il tcpwrapper e **sshd** il demone del servizio ssh. Viene inoltre indicata opzione **-i** per il demone **sshd**, indispensabile, poiche' indica che esso viene avviato tramite il superdemone **inetd**.

Venendo eseguito il demone **ssh** tramite **inetd** sono quindi stati rimossi tutti i link simbolici ad esso relativi nelle varie cartelle **/etc/rcX.d** con il comando **update-rc.d -f ssh remove** . L'opzione **-f** e' stata inserita per forzare la rimozione dei link simbolici, sebbene fosse ancora presente lo script **/etc/init.d/ssh** .

L'ultimo passo eseguito nella configurazione del wrapper tcp e' stata la modifica dei files **/etc/hosts.deny** ed **/etc/hosts.allow**, aggiungendo rispettivamente le seguenti righe :

```
#hosts.deny
sshd: ALL

#hosts.allow
sshd: 192.168.1.2 stargazer.localdomain
```

In questo modo **tcpd** filtra tutte le connessioni al servizio ssh, consentendo l'accesso ssh su **paranoid** esclusivamente dall'host **stargazer.localdomain** il cui indirizzo IP e' **192.168.1.2** .

Naturalmente perche' queste nuove impostazioni possano avere effetto e' necessario riavviare il superdemone **inetd** :

etc/init.d/inetd restart.

Infine, per aggiungere un ulteriore livello di filtraggio delle connessioni, se il kernel della macchina e' stato compilato con il supporto per **iptables**, e' possibile aggiungere la seguente regole ad **iptables**, in modo che netfilter accetti connessioni tcp sulla porta 22 esclusivamente quando provengono da stargazer:

```
iptables -A INPUT -p tcp --dport 22 -s 192.168.1.2 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
```

Completata la configurazione del servizio **ssh** e' stato creato il seguente shellscript **/usr/bin/rfc_create_db.sh** :

```
#!/bin/bash
HOST_NAME="paranoid"
aide --init
mv /var/lib/aide/aide.db.new /home/rfc/$HOST_NAME.aide.db
chmod 600 /home/rfc/$HOST_NAME.aide.db
chown rfc /home/rfc/$HOST_NAME.aide.db
chgrp rfc /home/rfc/$HOST_NAME.aide.db
```

Tale script crea un'immagine del stato attuale dei files del sistema tramite l'opzione **--init** di **aide**, e la sposta sotto la directory **/home/rfc**. In seguito imposta **rfc** come gruppo e proprietario del file, e setta i permessi in lettura e scrittura esclusivamente per il proprietario.

Questa scelta e' motivata dalla necessita' di effettuare una connessione via **ssh** (dall'host **stargazer** nel mio caso) da parte dell'utente **rfc**. Senza modificare opportunamente permessi e proprietario non sarebbe stato possibile accedere e copiare il database creato da **aide**.

Poiche' l'opzione **--init** di **aide** non puo' essere utilizzata da un semplice utente, ho consentito l'esecuzione dello script precedente tramite **sudo**, aggiungendo nel file **/etc/sudoers** di **paranoid**, la seguente riga:

```
rfc    paranoid = NOPASSWD: /usr/bin/rfc_create_db.sh
```

Come si puo' notare e' presente anche l'opzione **NOPASSWD**, per non richiedere l'immissione della password, poiche' il comando verra' richiamato all'interno dello shellscript **remotefc.sh** da **stargazer**.

Su **stargazer**, e' infatti presente il seguente shellscript :

```
#!/bin/bash
# ===== remotefc.sh =====
# ===== Remote File check == Andrea Beretta =====
# =====
# Thanks to : Remote Filesystem Checker di Claudio Panichi

if [ -z $2 ]; then
    echo "ERRORE : hostname non specificato"
    echo "Usage: $0 {update|get|compare} hostname"
    exit 1
fi
case "$1" in
    update)
        ssh rfc@$2 sudo rfc_create_db.sh
```

```

        cp /home/rfc/$2.aide.db /home/rfc/$2.db.bak.$(date +%d-%m-%Y)
        scp rfc@$2:/home/rfc/$2.aide.db /home/rfc
        ssh rfc@$2 rm -f /home/rfc/$2.aide.db
    ;;
get)
    ssh rfc@$2 sudo rfc_create_db.sh
    scp rfc@$2:/home/rfc/$2.aide.db /home/rfc/$2.aide.db.new
    ssh rfc@$2 rm -f /home/rfc/$2.aide.db
    ;;
compare)
    aide -c /home/rfc/$2.aide.conf --compare | mail -s " File
Integrity Check on $2" root@stargazer.localdomain
    ;;
*)
    echo "Usage: $0 {update|get|compare} hostname"
    exit 1
    ;;
esac
exit 0

```

Questo script puo' eseguire tre funzioni:

- **update:** viene effettuata una connessione via **ssh** come utente **rfc** verso l'host indicato dal secondo parametro dello script (**\$2**), e tramite **sudo** eseguito **rfc_create_db.sh** su tale host.
Una volta eseguito questo script, sulla macchina da cui si esegue **remotefc.sh** (nel mio caso **stargazer**), viene fatto un backup del precedente database, copiata la nuova versione tramite **scp**, e sempre via **ssh** rimosso dalla macchina remota (**paranoid**).
In questo modo nella directory **/home/rfc** dell'host **stargazer** compariranno i files **paranoid.aide.db** e **paranoid.db.bak.gg-mm-yy**. Ovviamente, se l'host remoto avesse avuto un hostname differente, anziche' **paranoid** il file avrebbe avuto tale hostname nella prima parte del nome.
Questo funzione deve essere eseguita dopo gli update e le modifiche della macchina monitorata.
- **get:** viene utilizzata per "scaricare" dall'host remoto l'immagine attuale dei files del sistema. Le operazioni eseguite sono pressoché identiche a quelle dell'opzione **update**, con l'unica differenza nella copia del file. Infatti, nella copia con **scp**, il file viene copiato con un nome differente (**nomehost.aide.db.new**) per non sovrascrivere il database precedente, già presente in **/home/rfc** (**nomehost.aide.db**) di **stargazer**.
- **compare:** esegue il confronto tra i due database **nomehost.aide.db** e **nomehost.aide.db.new**, sfruttando l'opzione **--compare** di **aide**. Viene inoltre usata una pipe, per inviare una mail all'amministratore di sistema, contenente il risultato del confronto.

Come si puo' notare dallo script, quando viene invocato **aide** su **stargazer** non viene utilizzato il file **/etc/aide.conf**, ma il file **hostname.aide.conf**. Lo script puo' infatti venire utilizzato per monitorare diverse macchine, e per ognuna di esse nella directory **/home/rfc** della macchina preposta al controllo devono essere presenti il file di configurazione ed i rispettivi database, distinguibili per il prefisso con l'hostname nel nome. Nel mio caso, controllando l'integrita' dei files dell'host **paranoid** il file di configurazione utilizzato e' appunto **paranoid.aide.conf**.

Tale file e' necessario solo per l'esecuzione del confronto, infatti la generazione dei database viene effettuata sulle macchine remote, e nel mio caso e' la copia del file **/etc/aide.conf** usato su **paranoid**, con l'unica differenza nei

percorsi ai file del database e del file di log:

```
database=file:/home/rfc/paranoid.aide.db
database_out=file:/home/rfc/paranoid.aide.db.new
database_new=file:/home/rfc/paranoid.aide.db.new
report_url=stdout
report_url=file:/home/rfc/log/paranoid.aide.log
```

Per eseguire il controllo dell'integrita' dell'host paranoid, quindi, debbono essere eseguite le seguenti operazioni:

- Creazione e copia del database iniziale dell'host **paranoid** con il comando **/home/rfc/remotefc.sh update paranoid**. Tale comando deve essere ripetuto ad ogni aggiornamento o modifica del software o dei files di configurazione della macchina, in modo da avere l'originale utilizzato per i confronti aggiornato.
Personalmente prima dell'update, io eseguirei comunque un confronto tra l'ultimo database originale e lo stato attuale del sistema. Se le differenze riscontrate sono solo quelle imputabili all'aggiornamento o alla modifica effettuata ai files di configurazione allora si puo' procedere alla sostituzione del database originale. In questo modo ci si puo' cautelare da eventuali modifiche tra i controlli.
- Download del database che rappresenta l'immagine attuale dell'host remoto con il comando **./home/rfc/remotefc.sh get paranoid**
- Confronto dell'ultimo database scaricato con l'originale, tramite **/home/rfc/remotefc.sh compare paranoid**.

Tutte le operazioni elencate sono state eseguite sull'host **stargazer** senza dovere ricorrere all'utente **root**, ma sempre con l'utente **rfc**. Durante la comparazione **aide** mostra a video alcuni messaggi di warning, legati ad alcuni privilegi mancanti, ma essi non vanno comunque ad inficiare l'operazione di confronto tra i due database.

L'ultimo passo da eseguire nella configurazione del sistema, e' l'inserimento delle operazioni di download e confronto dei database all'interno di un cronjob sull'host **stargazer**, automatizzando in questo modo l'esecuzione del tutto.

Ad esempio, per eseguire il controllo tutte le notti alle 3:15, l'utente **rfc** potrebbe inserire i seguenti cronjob con il comando **crontab -e**:

```
15 3 * * * /home/rfc/remotefc.sh get paranoid
25 3 * * * /home/rfc/remotefc.sh compare paranoid
```

Naturalmente l'utente **rfc** deve essere elencato nel file **/etc/cron.allow**.

Infine, per testare il sistema, infine, ho provato a creare un'immagine iniziale del sistema paranoid ed eseguire un confronto con il database scaricato dopo l'inserimento di un nuovo utente.

Come risultato di tale confronto nel file di log **/home/rfc/log/paranoid.aide.log** sono state riportate le seguenti informazioni, che evidenziano appunto l'avvenuta modifica dei files coinvolti nell'operazione di inserimento del nuovo utente.:

```
AIDE found differences between the two databases!!
Start timestamp: 2005-11-08 17:00:18
Summary:
Total number of files=2792,added files=0,removed files=0,changed files=9

Changed files:
```

```
changed:/etc
changed:/etc/passwd
changed:/etc/group
changed:/etc/passwd-
changed:/etc/shadow
changed:/etc/group-
changed:/etc/gshadow-
changed:/etc/shadow-
changed:/etc/gshadow
Detailed information about changes:
```

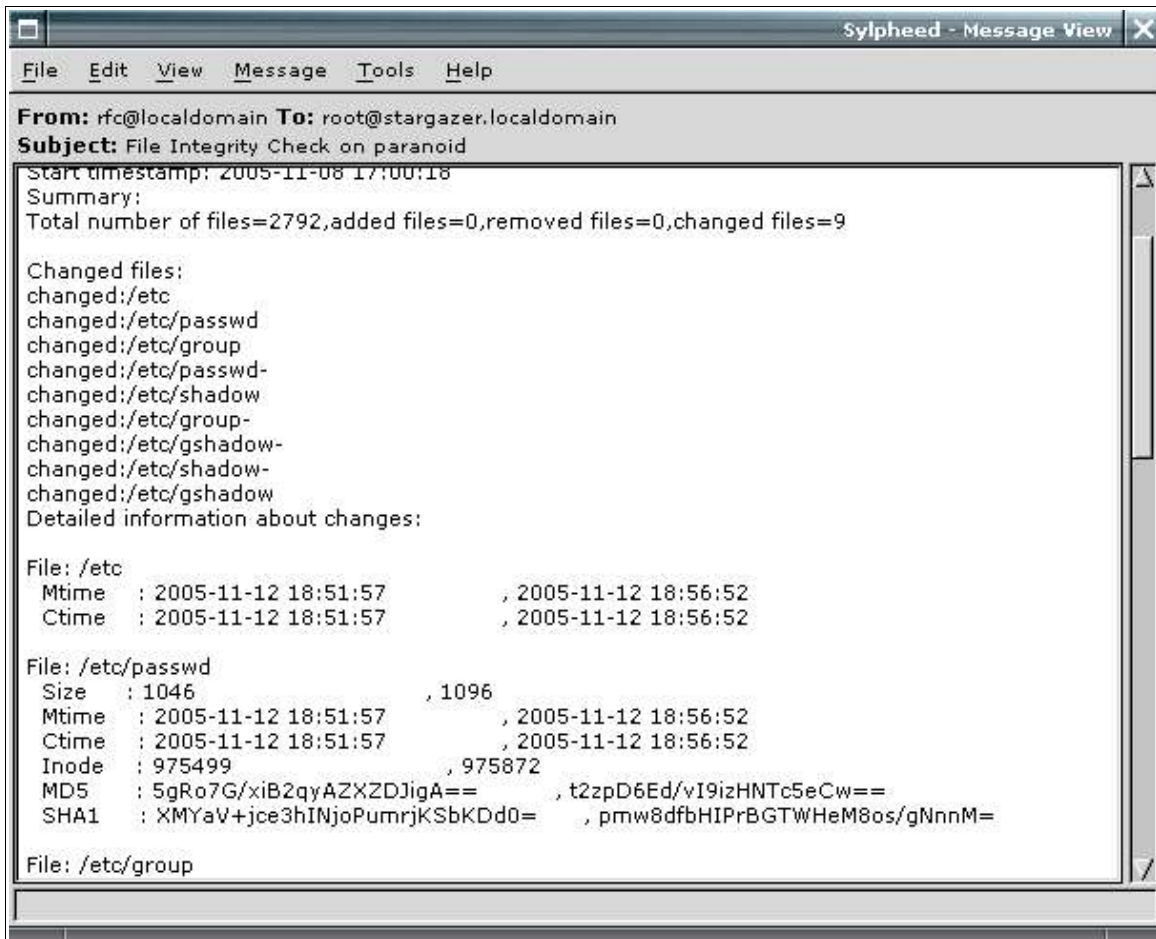
```
File: /etc
  Mtime      : 2005-11-12 18:51:57          , 2005-11-12 18:56:52
  Ctime      : 2005-11-12 18:51:57          , 2005-11-12 18:56:52
```

[...]

```
File: /etc/shadow-
  Size       : 745                          , 803
  Mtime      : 2005-11-11 19:28:25          , 2005-11-12 18:51:57
  Ctime      : 2005-11-11 19:31:12          , 2005-11-12 18:56:43
  MD5        : gFQqjCTXrRRmOj6fB0iZUg==    ,
qckc5NrpvPcd6nSmue7bpg==
  SHA1       : D0zAJY/qjzye0LMU60czES+ANNM= ,
R0BzpcRHmA9ggVoc2/n2d/YsnWc=
```

[...]

Inoltre, l'utente **root** dell'host **stargazer** ha ricevuto la seguente email dal subject **File Integrity Check on paranoid**, che riporta le medesime informazioni memorizzate nel file di log:



Utenti, gruppi e restrizione dei privilegi locali

Terminate le considerazioni relative al filesystem, e prima di iniziare a considerare altre parti quali il kernel, e' opportuno eseguire una serie di operazioni relative ad utenti e gruppi, cercando di limitarne anche i privilegi locali. All'interno di questa sezione, verranno inoltre indicate altre operazioni "spicciole" non inquadrabili negli altri capitoli della relazione.

Una prima operazione da eseguire e' disabilitare account e gruppi inutili. Molte distribuzioni includono infatti nel sistema diversi utenti e gruppi destinati a rimanere inutilizzati, sia per motivi "storici", che in previsione dell'installazione di servizi che li utilizzeranno. Per disabilitarli sarebbe possibile seguire due strade: eliminarli brutalmente tramite i comandi **deluser** e **delgroup**, oppure modificare opportunamente i files **/etc/shadow** ed **/etc/passwd**, in modo da poterli ripristinare rapidamente in caso di necessita'.

Ad esempio, dal contenuto del file **/etc/shadow**, si puo' notare come per tutti gli account diversi da **root** e da quelli aggiunti manualmente, ovvero **andrea**, **davide**, **rfc** e **standard**, il campo relativo alla password contenga il carattere * oppure !, in modo che non sia possibile fare il login con tali utenti :

```
root:$1$cHJ#.DlI/$i895yXlCnfPOiLMopPk8i/:10967:0:99999:7:::
daemon:!:10967:0:99999:7:::
bin:!:10967:0:99999:7:::
sys:!:10967:0:99999:7:::
sync:!:10967:0:99999:7:::
games:!:10967:0:99999:7:::
man:!:10967:0:99999:7:::
lp:!:10967:0:99999:7:::
mail:!:10967:0:99999:7:::
news:!:10967:0:99999:7:::
uucp:!:10967:0:99999:7:::
proxy:!:10967:0:99999:7:::
postgres:!:10967:0:99999:7:::
www-data:!:10967:0:99999:7:::
backup:!:10967:0:99999:7:::
operator:!:10967:0:99999:7:::
list:!:10967:0:99999:7:::
irc:!:10967:0:99999:7:::
gnats:!:10967:0:99999:7:::
nobody:!:10967:0:99999:7:::
andrea:$1$89JioLTw$OpNdRyBOlSoMQfUhlNoZXP0:10967:0:99999:7:::
postfix:!:10967:0:99999:7:::
sshd:!:13098:0:99999:7:::
rfc:$1$OVkNx1N3$g6EtS7Ue.6hjLeP7beN0A.:13099:0:99999:10:::
standard:!:13117:0:99999:10:::
davide:$1$SbMM5R3l$7dB0rFsvGstZjc0g4gt7BR:13120:0:99999:7:::
```

L'unico sistema per impersonare tali account sarebbe quindi l'utilizzo del comando **su** da parte dell'utente **root**, poiche' l'esecuzione di tale comando da parte di un utente normale, se consentita, richiederebbe comunque l'inserimento della password per l'utente che si vuole diventare.

Avendo pero' modificato opportunamente il file **/etc/shadow**, non sarebbe pero' possibile eseguire il confronto tra la password immessa e quella memorizzata nel file in questione.

Per impedire anche all'utente **root** di impersonare questi account bisognerebbe cambiare la shell loro assegnata con il comando **chsh -s /bin/false nomeutente**. Personalmente trovo questa seconda operazione superflua, poiche' solo **root** puo' impersonare questi utenti a cui e' impedito il login; se volesse, infatti, potrebbe benissimo cambiare nuovamente la shell dell'utente a cui e' interessato con un **chsh -s /bin/sh nomeutente**.

Altra operazione utile potrebbe essere quella di cercare di rendere piu' complesso il login dell'utente **root**, andando a modificare il file **/etc/securetty**. Tale file viene infatti letto dal programma **login** per sapere su quali consoles virtuali e' consentito l'accesso all'utente **root**.

Una prima possibilita' potrebbe essere la modifica di tale file indicando solo alcune consoles abilitate al login, anziche' tutte, ad esempio **tty3** e **tty6**:

```
#allowed consoles
tty3
tty6
```

Un sistema del genere ovviamente non puo' essere una misura di sicurezza particolarmente forte, ma costituisce comunque un fastidio in piu' per un eventuale attacker, poiche' dovra' anche provare le varie console virtuali fino a trovarne una abilitata.

Un'alternativa potrebbe essere quella rimuovere tutte le console virtuali definite in **/etc/tty**, in modo da non consentire il login diretto dell'utente **root**. Sarebbe dunque necessario loggarsi come un altro utente, e diventare successivamente **root** tramite il comando **su**, digitando due differenti password.

In un server dove sia stato disabilitato il **SUID** bit per il comando **su**, questa soluzione non sarebbe attuabile. Si potrebbe pensare di diventare **root** con il comando **sudo -s**, ma la password richiesta sarebbe quella dell'utente che ha invocato il comando.

Per richiedere la password di **root** si potrebbe modificare il file **/etc/sudoers**, aggiungendo la direttiva **runaspw**, che modificherebbe il comportamento di **sudo** richiedendo la password dell'utente con cui si vuole eseguire il comando. In questo modo, pero', tutti gli utenti che debbono eseguire comandi come **root** tramite **sudo** dovrebbero conoscerne la password.

Personalmente, lasciare il bit **SUID** attivo al comando **su** ritengo sia un rischio accettabile, se fatto con criterio, soprattutto nei confronti del beneficio di potere cosi' negare l'accesso diretto dell'utente **root** su qualsiasi tty. Anziche' modificare il file **/etc/securetty** ho pero' utilizzato il modulo **pam_access**, come verra' successivamente illustrato, poiche' esso consente la possibilita' di definire regole di accesso alle varie tty in maniera piu' granulare rispetto al vietare il **login** al solo utente **root**.

Altre operazioni consigliabili sono tutte quelle preposte al "tuning" degli account utente, che al solito consistono essenzialmente nella modifica di diversi files di configurazione. I files considerati sono stati per la precisione:

- **/etc/skel**: tale directory contiene tutti quei files, come ad esempio **.bashrc** e **.bash_profile**, che debbono essere copiati nelle home directory degli utenti all'atto della loro creazione. E' quindi opportuno definire tali files in modo che la "configurazione" di default per i nuovi utenti creati sia quella ottimale ai propri scopi.

All'interno di **skel** sono intervenuto su tali files :

- **.bashrc** : e' stato verificato che la variabile d'ambiente **PS1** fosse cosi' impostata : **PS1=[\"u@\\h \\w\"]\$** , in modo che il prompt shell dell'utente mostri il nome dell'utente, il nome dell'host e la directory corrente.

Ad esempio se l'utente andrea si trovasse in **/etc/skel** il prompt sarebbe il seguente **[andrea@paranoid:/etc/skel]\$** . Penso sia utile mostrare il nome dell'utente e la directory corrente, in modo da non dovere richiamare il comando **pwd** ogni volta che si desideri sapere in quale punto del filesystem ci si trovi.

Altra modifica da apportare potrebbe essere la creazione di alcuni alias quali **ls**, **rm**, **cp** ed **mv**:

```
alias ls='ls --h -color=auto'
alias cp='cp -i'
alias rm='rm -i'
alias mv='mv -i'
```

Il primo alias ha uno scopo prettamente estetico abilitando l'utilizzo dei colori nell'output del comando **ls**, in modo da distinguere immediatamente directory, files, link, eseguibili, ecc. In base ai loro colori. Viene inoltre specificata l'opzione **-h**, per mostrare le dimensioni dei file in maniera piu' leggibile, ovvero in kb, mb ecc. anziche' in bytes.

Gli altri tre alias invece fanno in modo che venga sempre chiesta la conferma all'utente prima di cancellare o sovrascrivere un file gia' esistente. In tal modo DOVREBBE essere meno probabile che egli possa incorrere in una cancellazione accidentale, dovendo anche rispondere affermativamente alla richiesta di conferma dell'operazione.

Sono stati infine aggiunti due ulteriori alias per le operazioni di decompressione degli archivi di tipo **tar.gz** e **tar.bz2**. La cosa non ha alcuna rilevanza ai fini della sicurezza, ma e' comunque una maggiore comodita' anziche' dovere ogni volta specificare un comando con quattro opzioni:

```
alias untargz='tar -zxvf'
alias untarbz2='tar -jxvf'
```

- **.bash_logout**: definisce i comandi e le operazioni da eseguire al logout dell'utente. Una soluzione abbastanza paranoica, che non viene implementata di default potrebbe essere la cancellazione della bash history dell'utente al momento del suo logout e la "pulizia" dello schermo, per evitare che chi passi nei pressi del monitor possa vedere gli ultimi comandi eseguiti dallo user precedente. A tal fine e' stato creato il file **.bash_logout** con il comando **touch /etc/skel/.bash_logout**, aggiungendovi le seguenti righe :

```
#cancellazione history
cat /dev/null > $HOME/.bash_history
#pulizia terminale
clear
```

- **.bash_profile**: all'interno di questo file viene normalmente impostata la variabile d'ambiente **PATH** ed il valore del parametro **umask**. La variabile **PATH** definisce le directory entro cui cercare gli eseguibili quando viene indicato un comando in shell, e viene impostata al valore **"/usr/local/bin:/usr/bin:/bin"**, eliminando rispetto all'impostazione di default le directory **/usr/bin/X11** e **/usr/games**.

Un altro parametro estremamente importante e' il valore di **umask**, che indica i permessi di default con cui vengono creati files e directory. Personalmente preferirei che solo il proprietario di un file possa accedervi per default, ed in caso gli sia concesso possa cambiarne i permessi in caso di necessita'. Per questo motivo e' stato imposto un valore di **umask** pari a **0077**, che equivale a creare files con privilegi di accesso impostati per default a **700**.

- **/etc/adduser.conf**: definisce il comportamento del comando **adduser**, quindi consente di impostare diversi parametri relativi alla creazione di nuovi account:
 - **USERGROUPS**: impostando a no questa opzione e' stato fatto in modo che ogni nuovo utente appartenga al gruppo con il gid specificato dall'opzione **USERS_GID**. Nel caso il valore di

USERS_GID e' uguale a 100, e fa riferimento al gruppo **users**. L'impostazione di default imponeva che per ogni utente venisse creato un gruppo omonimo, ma ho preferito che tutti i nuovi utenti appartenessero ad uno stesso gruppo, in modo da potere definire in modo piu' semplice le regole sull'utilizzo delle risorse in **/etc/security/limits.conf**.

- **DIR_MODE** definisce i permessi dell'home directory dell'utente, ed e' impostato per default al valore **0755**, consentendo quindi permessi di lettura ed esecuzione anche per tutti gli altri utenti del sistema. Volendo invece che le directory ed i files in esse contenuti fossero accessibili esclusivamente ai rispettivi proprietari e' stato cambiato a **0700** il valore di questa opzione.
- **QUOTA_USER** definisce l'utente standard per l'assegnazione delle quote sul filesystem. Anziche' definire manualmente le quote per ogni utente, sarebbe piu' opportuno prevedere dei valori standard, ed al limite modificarli solo per gli utenti con particolari esigenze.

Per tale scopo e' stato creato un utente chiamato **standard** con il comando **adduser standard**, e si sono editate le sue quote disco con **edquota -u standard**, imponendo limiti piuttosto restrittivi come ad esempio 40 Mb di quota soft e 60 di quota hard.

Così facendo alla creazione di ogni nuovo utente, il sistema provvedera' a copiare i valori delle quote di **standard** eseguendo automaticamente il comando **edquota -p standard**.

Poiche' l'utente **standard** deve essere utilizzato solo come "modello" per le quote disco, si e' preferito non assegnargli alcuna shell e negargli il login, modificando le righe ad esso relative nei files **/etc/passwd** ed **/etc/shadow**.

Per la precisione in **/etc/shadow**, si e' sostituito il campo contenente la cifratura della password con un asterisco :

```
standard:*:13117:0:99999:7: : :
```

In **/etc/passwd** invece e' stato rimosso il riferimento alla home directory dell'utente ed impostata la sua shell di login a **/bin/false**:

```
standard:x:1003:1003, , , : :/bin/false
```

Teoricamente sarebbe stato sufficiente impostare solamente la shell a **/bin/false**, che avrebbe comunque consentito il login all'utente, ma lo avrebbe immediatamente sloggato dopo la comparsa del motd. Personalmente ho preferito modificare anche **/etc/shadow** per negare completamente l'accesso.

E' stata infine rimossa l'home directory di **standard** con un **rm -rf /home/standard**.

- **/etc/login.defs**: definisce una serie di controlli e di opzioni per il programma **login**. Tra le opzioni piu' importanti :
 - **FAIL_DELAY**: specifica l'arco di tempo, espresso in secondi, prima che possa essere consentito un tentativo di login in seguito ad un fallimento. Il valore di default sarebbe 3 secondi, ma ho preferito impostarlo a 10. In caso di errore di battitura della propria password, un utente legittimo puo' comunque permettersi di aspettare 10 secondi, mentre un'attacker che tenti un attacco di tipo brute force sarebbe rallentato notevolmente con un attesa di 10 secondi tra un tentativo e l'altro
 - **FAILLOG_ENAB**: e' stato impostato a **yes** in modo da abilitare il logging in **/var/log/faillog** dei tentativi di login falliti.
 - **LOG_OK_LOGINS**: indica se abilitare o meno il logging dei login avvenuti con successo. Per default viene impostata a no, ma ho ritenuto utile cambiare tale opzione in **yes** per tenere traccia anche degli utenti che sono correttamente acceduti al sistema, poiche' potrebbe costituire un indizio o un elemento di

correlazione in caso di problemi.

- **SULOG_FILE**: decommentando questa riga si abilita il logging dell'attivita' del comando **su**, nel file indicato come argomento (nel mio caso **/var/log/sulog**). Per default questa opzione non e' abilitata e tale attivita' viene memorizzata nei comuni log; scelta opinabile dal mio punto di vista perche' rende meno agevole l'analisi dei log stessi.
 - **FTMP_FILE**: permette di loggare i tentativi di login falliti nel file indicato come argomento, ovvero **/var/log/btmp**, in modo da poterli visionare agevolmente con il comando **lastb**.
 - **PASS_MAX_DAYS**: definisce il massimo numero di giorni di validita' di una password. Se una password dovesse risultare piu' vecchia di tale valore, al prossimo login verrebbe imposto un cambio della parola chiave all'utente. Il valore scelto per questo parametro e' stato di **180** giorni anziche' il default 99999
 - **PASS_MIN_DAYS**: indica il numero minimo di giorni che debbono trascorrere tra due cambi di password. Ho ritenuto opportuno mantenere il valore di default **0**.
 - **PASS_WARN_AGE**: indica per quanti giorni prima della scadenza della password l'utente deve essere avvisato. Anziche' il valore di default 7 ho impostato un valore di **10** giorni. (NOTA: **PASS_MAX_DAYS**, **PASS_MIN_DAYS** e **PASS_WARN_AGE** vengono utilizzati solo alla creazione dell'utente, quindi, modifiche a questi valori non influenzeranno gli account preesistenti, per i quali sara' necessario ricorrere al comando **chage**)
 - **DEFAULT_HOME**: se impostato a no, non consente il login ad utenti per cui non sia possibile accedere alla relativa home indicata in **/etc/passwd**. Il valore di default era impostato a **yes**, assegnando quindi in questi casi la directory **/** come home. Ho preferito impostarla **no**, in modo che un utente privo di home directory non abbia possibilita' di accedere.
- **/etc/security/limit.conf** : regola l'assegnazione delle risorse agli utenti del sistema, in base alla definizione di regole che consentono di impostare dei limiti specifici. Nel mio caso ho impostato dei limiti per gli utenti appartenenti al gruppo **users**, ovvero il gruppo a cui vengono assegnati per default i nuovi utenti creati con il comando **adduser**.

Per la precisione i limiti impostati sono stati i seguenti :

- **0 mb** per la dimensione di un file di core. Viste le quote abbastanza restrittive gia' imposte agli utenti, consentire al kernel la creazione di immagini dei processi terminati inaspettatamente in un file di core limiterebbe ulteriormente lo spazio a disposizione degli utenti. Naturalmente la dimensione da associare ai file di core deve essere valutata a seconda dei casi.
Ad esempio se sul server in questione si stesse testando un software ancora in fase di sviluppo, disabilitare i file di core non penso sarebbe un'idea particolarmente felice, poiche' potrebbero dare informazioni utili ai fini del debugging.
Inoltre, ritengo che ad esclusione di casi particolari, quali appunti degli sviluppatori, sia meglio non consentire la creazione e visione di file core con l'immagine in memoria del processo, poiche' potrebbero fornire informazioni utili per lo sfruttamento di eventuali vulnerabilita' del programma in questione.
- **70 Mb** per tutti i limiti relativi alla memoria ram che l'utente e' abilitato ad utilizzare, quindi massima dimensione dello stack e massimo spazio di indirizzamento. Il valore impostato e' piu' che altro un esempio, poiche' non saprei dare una regola fissa in base a cui stabilire questo valore. Nella scelta andrebbero considerate la quantita' di memoria ram presente sul server, gli utenti connessi, gli applicativi che debbono girare.
Un buon sistema potrebbe essere quello di osservare tramite il comando **top** quali siano i processi in esecuzione che normalmente consumano maggiore memoria e da chi vengono normalmente eseguiti.
- **60 mb** per la massima dimensione di un file. Ho impostato tale valore pari alla quota hard per il

filesystem. In caso di quote disco particolarmente generose, potrebbe essere una buona idea imporre **fsize** ad un valore piu' restrittivo rispetto alla quota.

In caso contrario, un utente con cattive intenzioni potrebbe creare volutamente un file di dimensioni particolarmente elevate, e continuare a copiarlo e cancellarlo, ad esempio tramite uno script. Una situazione del genere non sarebbe certamente un toccasana per il disco e in generale le prestazioni di I/O del sistema.

- **60 mb** di quota hard, e **40 mb** di quota soft per la massima dimensione dei dati. Come si puo' notare sono gli stessi valori delle quote precedentemente impostate per il filesystem.
- **256** come massimo numero di file aperti. Il valore di default sarebbe 1024, come si puo' vedere dal comando **ulimit -n**, ma ho preferito essere piu' restrittivo.

In caso di problemi o necessita' particolari e' comunque possibile imporre nuovi limiti anche per singoli utenti, anziche' in base al gruppo di appartenenza, in modo da potere regolare tutte le singole eccezioni.

Nel caso diversi utenti dovessero aprire troppi files contemporaneamente, il sistema potrebbe ritornare un errore, relativamente al fatto che esistono troppi files aperti. Il numero massimo di file apribili dal sistema, nel mio caso era impostato a 8192 (verificabile con un **cat /proc/sys/fs/file-max**).

Una regola di base per la scelta di tale valore e' quella di consentire l'apertura di 256 files ogni 4 Mb. di ram. Avendo 128 mb di ram su tale macchina, 8192 sembrerebbe dunque il valore corretto.

Secondo diverse fonti, pero', nel caso di webserver, o mailserver sarebbe opportuno incrementare tali valori. Nel mio caso ho verificato, osservando il contenuto di **/proc/sys/fs/file-nr**, che il limite era decisamente superiore al numero di files aperti (8192 contro 247), ed ho quindi lasciato tale parametro invariato.

In caso si dovessero osservare diversi messaggi di errore tipo **running out of open files**, sara' opportuno aumentare questo limite: ad esempio, volendo impostare il limite a 32768 files, e' necessario modificare il valore in **/proc/sys/fs/file-max**, oppure aggiungere la riga **fs.file-max=32768** in **/etc/sysctl.conf**.

Inoltre, la manpage del filesystem proc consiglia anche di modificare il massimo numero di **inodes**, che andrebbe impostato ad un valore pari a 3-4 volte **file-max**. Secondo la guida di riferimento di debian e mailing list relative al kernel Linux, pero', a partire dalla release **2.4.18** il numero di inodes dovrebbe essere allocato dinamicamente, quindi sarebbe sufficiente impostare esclusivamente il valore di **file-max**.

- **1** come massimo numero di login contemporanei concessi all'utente. Trattandosi di utenti normali, e non amministratori di sistema, o sviluppatori, ho preferito consentire un unico login.
- **40** come massimo numero di processi eseguibili contemporaneamente. Ho provato ad eseguire il comando **ps -u andrea**, su una macchina con Xserver, dove erano in esecuzione diverse applicazioni tra cui **firefox**, **openoffice**, l'interprete **python**, **xmms**, ecc, per farmi un'idea di quanti processi potessero essere attivi per l'utente in questione.

Sono risultati attivi 14 processi. Sul server preparato per la relazione naturalmente non e' presente alcun server X, e neppure alcuno dei diversi applicativi menzionati. Volendo stare largo ho quindi imposto un limite di **40**.

La scelta di imporre un limite al numero dei processi e' dovuta principalmente al timore di tentativi di Denial Of Service da parte degli utenti. Senza imporre tale limite, ad esempio, sarebbe sufficiente una banalissima forkbomb per paralizzare il sistema. A titolo di prova ho copiato sulla macchina l'eseguibile compilato dal seguente frammento di codice in C :

```
#include <unistd.h>
void main(void)
{
    while(1)
        fork();
}
```

}

Eseguendolo con un utente del gruppo **users**, ad esempio l'utente **rossana**, ho potuto verificare tramite il comando **ps -u rossana | wc -l** eseguito in un'altra tty, che il numero totale di processi in esecuzione era appunto 40. Per la precisione il numero di linee contate da **wc** era 41, comprendendo però anche la prima linea del comando **ps**.

Eseguendolo invece con un altro utente, ad esempio **andrea**, il sistema diventava pressoché inutilizzabile e per nulla responsivo, finché non si è eseguito un **ctrl+c** nella console da cui si era eseguita la **forkbomb**.

Nello specifico caso del mio server non essendo presente il compilatore C, ed avendo attivato l'opzione **noexec** per la partizione montata sotto **/home**, non sarebbe stato possibile per un utente normale portare a termine un simile DoS, però l'esempio mi è sembrato utile a sottolineare l'importanza di imporre un limite anche al numero dei processi per utente.

Riassumendo il file **/etc/limits.conf** preparato è stato il seguente :

```
#max size per un file core
@users      hard   core           0
#limiti in kb. per massimo utilizzo memoria
@users      hard   memlock        71680
@users      hard   stack          71680
#max size per un file
@users      hard   fsize          61440
#quote disco
@users      soft   data            40960
@users      hard   data            61440
#limite nr. files aperti contemporaneamente
@users      hard   nofile          256
#login contemporanei consentiti
@users      hard   maxlogins       1
#limite nr. Processi contemporanei
@users      hard   nproc           40
```

- **/etc/security/time.conf**: consente di stabilire delle regole di accesso in base ai giorni della settimana ed all'orario. Dettagli sul suo impiego verranno mostrati trattando il modulo **pam_time**.
- **/etc/security/access.conf**: consente di stabilire regole di accesso in base a username, gruppi, numero di tty ecc. Maggiori dettagli sul suo impiego verranno mostrati trattando il modulo **pam_access**.

Infine, ulteriore operazione eseguita è stata la creazione di due gruppi: **administrators** e **sys_operators** con i comandi **groupadd administrators** e **groupadd sys_operators**. Ai due gruppi menzionati ho aggiunto rispettivamente gli utenti **andrea** e **davide** con **gpasswd -a andrea administrators** e **gpasswd -a davide sys_operators**.

L'utilità della creazione di questi due gruppi è legata alla necessità di non imporre i limiti previsti per il gruppo **users** anche agli utenti di questi due gruppi. Naturalmente ho provveduto anche a rimuovere gli utenti **andrea** e **davide** dal gruppo **users**, altrimenti tale accorgimento sarebbe risultato del tutto inutile.

In alternativa, anziché definire i gruppi **administrators** e **sys_operators**, sarebbe stato possibile creare due gruppi

omonimi del nome utente, ma questa scelta avrebbe potuto risultare scomoda nella definizione delle regole in /
etc/access.conf, come verra' illustrato nella sezione successiva trattando **PAM**.

Pluggable Authentication Modules

Pluggable authentication modules (**PAM**) sono un insieme di librerie che consentono ad un amministratore di sistema di definire con una certa flessibilit , come le applicazioni debbano autenticare gli utenti.

Il funzionamento di questo sistema prevede la presenza all'interno della directory **/etc/pam.d** di un file per ogni applicazione per la cui autenticazione si desidera usare **PAM**. All'interno di tali files si trova la sequenza, o "stack di autenticazione", delle condizioni che debbono essere soddisfatte per autenticare l'utente. Nelle regole definite dallo stack possono essere richiamati diversi moduli con funzioni specifiche.

Se per un servizio od un programma per cui si necessita di autenticazione, non esiste il rispettivo file in **/etc/pam.d** il sistema fara' riferimento ad una configurazione di default definita in **/etc/pam.d/other**.

Questa relazione non vuole essere una guida di riferimento dettagliata per il sistema **PAM**, quindi anziche' spiegare sintassi varie e dettagli teorici del funzionamento, ho preferito fare alcuni esempi specifici del suo utilizzo, concentrandomi sulla loro spiegazione e sul perche' possano essere utili ai fini del miglioramento della sicurezza del sistema.

- Personalmente, ritengo che il primo aspetto da considerare sia la configurazione di **/etc/pam.d/other**, cercando di renderla il piu' restrittiva possibile. Per default, l'installazione di programmi che utilizzino **PAM** comprende anche la creazione del file ad esso relativo in **pam.d**, quindi l'utilizzo dello stack di autenticazione definito in **other**, rappresenta l'eccezione. Per questo motivo ritengo esso debba essere particolarmente restrittivo, poiche' verrebbe utilizzato per situazioni non definite a priori.

Il contenuto previsto per tale file e' quindi stato:

auth	required	pam_deny.so
auth	required	pam_warn.so
account	required	pam_deny.so
account	required	pam_warn.so
password	required	pam_deny.so
password	required	pam_warn.so
session	required	pam_deny.so
session	required	pam_warn.so

Tutti gli elementi dello stack hanno flag di controllo **required**, ovvero l'azione del modulo deve andare a buon fine, oppure la richiesta del servizio fallira' dopo l'elaborazione della parte residua dello stack.

Per tutti i quattro contesti **auth**, **account**, **password** e **session** vengono richiamati i moduli **pam_deny** e **pam_warn**. Il primo di essi non fa altro che negare l'accesso alla risorsa, mentre il secondo registra via **syslog** l'utente, l'host di provenienza, il servizio richiesto ecc. Questa configurazione, quindi, non consentira' alcun tipo di accesso, oltre a loggare i tentativi effettuati.

Bisogna pero' tenere conto che in caso di errore da parte dell'amministratore di sistema, una configurazione cosi' restrittiva potrebbe avere effetti molto fastidiosi. Se ad esempio l'amministratore dovesse cancellare per errore **/etc/pam.d/login**, il programma **login** non potendo piu' fare riferimento a tale file, considererebbe **/etc/pam.d/other**, con conseguente impossibilit  di accedere al sistema.

- Rafforzare la politica di definizione delle password: sebbene non si sia ancora toccato questo argomento e' scontata l'importanza di utilizzare delle password sicure, ovvero abbastanza complicate da essere indovinate o scoperte con attacchi basati su dizionario. Naturalmente con password esageratamente complesse, l'utente tendera' ad annotarle, nella peggiore delle ipotesi su un post-it attaccato sotto la tastiera. Una situazione del

genere sarebbe dannosa quanto l'utilizzo di password troppo deboli. Quindi sarebbe opportuno trovare una via di mezzo tra le due situazioni.

Fornire esclusivamente istruzioni e consigli agli utenti su come debbano definire le proprie password, credo sia pressoché inutile, poiché avendone la possibilità essi continuerebbero ad usare password semplici quali nomi di persona, date di nascita, squadra del cuore e così via.

E' quindi necessario obbligare gli utenti a scegliere password adeguate, tramite l'utilizzo di **PAM**, appunto. Un modulo fondamentale per tale scopo è **pam_cracklib**.

Tale modulo non è presente al termine dell'installazione base del sistema, e quindi è necessario installarlo tramite il comando **apt-get install libpam-cracklib**. Tra le dipendenze viene installato anche un dizionario, che potrà essere utilizzato per impedire la scelta di password uguali alle parole che lo compongono. Per qualche oscuro motivo, però, anziché un dizionario inglese od italiano ne viene installato uno in catalano.

Si è quindi provveduto a rimuoverlo con il comando **apt-get remove wcatalan**, ed al suo posto ho installato i dizionari italiano, inglese ed americano con **apt-get install witalian wamerican wenglish**.

Perché il modulo **pam_cracklib** potesse farvi riferimento ho editato il file **/etc/cracklib.conf**, modificando nel modo seguente la riga **cracklib_dictpath_src**:

```
cracklib_dictpath_search="/usr/share/dict/italian
/usr/share/dict/american-english /usr/share/dict/british-english
/usr/share/dict/extra.words"
```

L'opzione modificata consente di specificare dei files contenenti delle liste di parole, che potranno essere aggiunti al database della libreria **cracklib** con il comando **update-cracklib -a -r /etc/cracklib/cracklib.conf**

La scelta dell'utilizzo di dizionari di diverse lingue ritenga sia giustificata dalla necessità di impedire che gli utenti possano utilizzare parole dei dizionari inglese ed americano come password. Volendo essere veramente paranoici si potrebbero aggiungere anche altri dizionari quali quello francese, tedesco ecc.

Un'ulteriore precauzione presa, è stata la creazione del file **/usr/share/dict/extra.words**, anch'esso richiamato in **cracklib.conf**, al cui interno andranno definite tutte quelle parole non appartenenti al vocabolario, ma che potrebbero essere facilmente utilizzate come password, come appunti nomi di squadre di calcio, personaggi dei fumetti ecc.

Il funzionamento del modulo **pam_cracklib** precedentemente menzionato è basato sulla libreria **crackLib**, per controllare le password e capire se rispettano determinati criteri di sicurezza. Tra i controlli effettuati dal modulo si hanno:

- corrispondenza con parole presenti nel dizionario;
- verifica della presenza di pattern troppo semplici;
- controllo che le password non siano parole generabili a partire dalle finger information riportate in **/etc/passwd**;
- verifica che la nuova password non sia la precedente scritta al contrario
- verifica che la nuova password non sia la precedente in maiuscolo o viceversa;

La politica prescelta per la definizione delle password è stata quindi la seguente:

- lunghezza di almeno 8 caratteri;
- hash **md5** della password in **/etc/shadow**;
- presenza di almeno una lettera maiuscola;
- presenza di almeno un numero;
- verifica che la parola non appartenga al vocabolario;

Per ottenere questo comportamento e' stato modificato nel modo seguente il file `/etc/pam.d/passwd`:

```
password    required    pam_cracklib.so    dcredit=-1 ucredit=-1
minlen=8
password    required    pam_unix.so        obscure md5
use_authtok
```

Come si puo' notare entrambe le regole riguardano il tipo di modulo **password**, ovvero quello preposto all'aggiornamento dei token di autenticazione relativi all'utente. Poiche' si vuole che entrambe le regole vengano onorate il flag di controllo e' posto a **required**, in modo che la richiesta del servizio (nel caso il programma **passwd**) fallisca in caso l'azione di almeno uno dei modulo non abbia l'esito richiesto.

Nella prima regola viene richiamato il modulo **pam_cracklib** con i parametri **dcredit** ed **ucredit** pari a **-1** e **minlen** uguale a **8**. Tale regola richiede che la lunghezza della password sia di almeno 8 caratteri e contenente almeno un numero ed una lettera maiuscola.

I valori negativi dei parametri **dcredit** ed **ucredit** definiscono appunto i primi due requisiti, disabilitando pero' il sistema di attribuzione dei crediti alla password. In caso contrario, abilitando il sistema di crediti per ogni numero e ogni lettera maiuscola verra' calcolato un credito pari rispettivamente al valore di **dcredit** ed **ucredit**. In tale caso la password verrebbe accettata se la sua lunghezza fosse pari almeno alla differenza tra **minlen** e la somma di questi crediti.

La seconda regola invece utilizza il modulo **pam_unix**. Il parametro **obscure** dovrebbe servire per eseguire una ulteriore serie di controlli, non specificati pero' nella documentazione; **md5** indica di memorizzare le password usando un hash md5 anziche' la funzione **crypt**, ed infine **use_authtok** obbliga **pam_unix** ad impostare automaticamente come nuova password quella gia' accettata dal precedente modulo nello stack (nel caso **pam_cracklib**). Infatti, non utilizzando quest'ultima opzione, ho avuto modo di verificare che all'utente verrebbe chiesto di cambiare la password due volte, la prima tramite il modulo **pam_cracklib** e la seconda tramite il modulo **pam_unix**.

Come si puo' notare, inoltre, non ho impostato l'opzione **nullok**, che avrebbe consentito il cambio di password anche nel caso la password attuale fosse stata vuota. In caso dovesse presentarsi tale necessita', l'utente **root** sarebbe comunque in grado di effettuare il cambio, poiche' queste regole non debbono essere onorate nel suo caso.

Per verificare il funzionamento della politica adottata ho provato ad impostare alcune password prive di numeri o lettere maiuscole, troppo corte, troppo ripetitive (come ad esempio `asdasdasd`) o appartenenti al vocabolario. Come previsto il sistema ha impedito la modifica, riportando rispettivamente i seguenti avvisi: **BAD PASSWORD: is too simple**, **BAD PASSWORD: is too short**, **BAD PASSWORD: it does not contain enough different characters**, e **BAD PASSWORD: it is based on a dictionary word**.

- Impedire il riutilizzo di password precedenti. Un'altra caratteristica utile nella politica di definizione delle password e' anche quella di impedire il riutilizzo di password usate in precedenza, poiche', in caso contrario buona parte degli utenti, per pigrizia, userebbero sempre le solite due o tre password cambiandole ciclicamente.

Tramite il modulo **pam_unix** e' pero' possibile memorizzare le ultime password degli utenti nel file `/etc/security/opasswd`, in modo che al cambio di password si possa controllare che la parola scelta non sia compresa in questo elenco. Il numero di password memorizzate puo' essere impostato con il parametro **remember**.

Non esistendo il file **opasswd** e' stato creato con un `touch /etc/security/opasswd`, pena il non funzionamento del parametro **remember**. Avendo impostato in precedenza un parametro **umask** piuttosto restrittivo, inoltre, non e' stato necessario modificare i permessi del file perche' fosse leggibile e scrivibile solo da **root**.

Permessi piu' "generosi", ad esempio consentendo la lettura a chiunque, costituirebbero un punto debole ai fini

della sicurezza, poiche' tutti potrebbero accedere a tale file. Certamente le password ricavate, con qualche programma quale John The Ripper ad esempio, non sarebbero piu' valide ma potrebbero comunque fornire informazioni utili sul meccanismo che gli utenti scelgono per definire le loro password.

Anche con questo accorgimento, pero', la pigrizia dell'utente potrebbe ancora avere il sopravvento, ad esempio cambiando solo una lettera od un numero dalla password precedente. Per impedire un comportamento simile si potrebbe specificare il numero minimo di caratteri diversi che la nuova password deve contenere rispetto alla precedente, tramite il parametro **difok** del modulo **pam_cracklib**.

Personalmente ho scelto di impedire il riutilizzo delle ultime 20 password, ed obbligare gli utenti a cambiare almeno tre caratteri nelle nuove parole chiave, quindi ho modificato il file **/etc/pam.d/passwd** aggiungendo i **parametri difok=3 e remember=20** :

```
password    required    pam_cracklib.so    dcredit=-1 ucredit=-1
minlen=8 difok=3
password    required    pam_unix.so        obscure md5
use_authtok remember=20
```

- Locking degli account: l'idea e' quella di bloccare l'account di un utente dopo alcuni tentativi consecutivi di login errati. Sono consapevole che questa caratteristica potrebbe prestare il fianco ad attacchi DoS, o scherzi idioti quali il cercare di loggarsi con lo username di qualche collega ed una password errata per bloccarne l'account. Questa scelta ha pero' l'indubbio vantaggio di rendere impossibile tentativi esaustivi per indovinare la password di un'utente, avendo solamente un numero limitato di tentativi.

Il problema degli attacchi DoS, e' invece legato agli account di sistema utilizzati per i servizi, dai quali e' pero' possibile mettersi al riparo, disabilitando il login diretto per tali account, come verra' illustrato successivamente.

Nel mio caso ho quindi scelto di bloccare solo gli account utente, per la precisione dopo 5 errori consecutivi, inserendo le due righe seguenti in **/etc/pam.d/login**:

```
auth        required    pam_tally.so        onerr=fail unlock_time=20
deny=5 per_user
account      required    pam_tally.so        reset
```

La prima riga, relativa all'autenticazione, consente di contare il numero dei tentativi di login o di su falliti per ogni utente, andando a leggerli dal file **/var/log/faillog**. L'opzione **onerr=fail** indica al modulo come comportarsi in caso di errore, come ad esempio l'impossibilita' di accedere al file **faillog**, mentre **unlock_time=20** indica al sistema di bloccare l'account dell'utente per 20 minuti in caso di un numero di tentativi di login errati superiore al valore di **deny**.

La seconda riga gestisce altri aspetti dell'account, resettando il contatore dei login falliti in caso l'utente sia stato autenticato con successo.

Anziche' bloccare definitivamente l'account, richiedendo quindi l'intervento dell'amministratore di sistema in caso qualche utente avesse inavvertitamente sbagliato troppe volte ad inserire la propria password (ad esempio per non essersi accorto del caps lock attivo), ho preferito fare in modo che esso resti inaccessibile per 20 minuti. Ritengo che per qualcuno che voglia fare delle prove esaustive per scoprire una password, questo deterrente possa essere sufficiente.

Inoltre, temendo che qualcuno possa cercare di bloccare alcuni account di sistema (**NOTA** la possibilita' di login per questi account dovrebbe essere disabilitata, ma preferisco essere un po' paranoico), nella prima riga e' stato utilizzato il parametro **per_user**. Tale parametro indica al modulo **pam_tally** di ignorare il valore di **deny** per gli account per i quali il massimo numero di errori al login sia stato settato manualmente, ad esempio con il comando **faillog -u nomeutente -m numero_errori**. Impostando il valore di **numero_errori** a -1 si puo'

disattivare il limite al numero di login falliti, mentre il valore zero indica di usare come riferimento il parametro **deny** del modulo **pam_tally**.

In caso di necessita' e' inoltre possibile verificare il numero di fallimenti associati a ciascun utente con il comando **pam_tally nomeutente**, ed eventualmente riportarli a zero con **pam_tally --user nomeutente --reset**.

Inoltre, si ricorda che per default il modulo **pam_tally** non incrementa il contatore dei login falliti nel caso dell'utente **root**, perche' in caso contrario sarebbe troppo semplice causare un problema al sistema bloccando tale account. Volendo e' possibile modificare questo comportamento tramite l'opzione **even_deny_root**, ma ho preferito lasciare l'impostazione di default, considerando maggiori i rischi in cui si potrebbe incorrere rispetto ai benefici derivanti dall'opzione menzionata.

- Impedire il login diretto per gli account di sistema ed account condivisi. Ai fini di una migliore sicurezza del sistema, ritengo sia necessario impedire il login diretto per l'utente **root**, eventuali account di sistema che non siano stati disattivati ed account condivisi (ovvero la cui password sia conosciuta a piu' persone). In questo modo sara' prima necessario effettuare il login con un altro utente e poi assumere le credenziali desiderate tramite **su**.

Per ottenere questo comportamento e' stato fatto ricorso al modulo **pam_access**, che consente di definire delle regole in **/etc/security/access.conf** in maniera simile a come si fa per il tcwrapper in **hosts.allow**.

E' quindi stato sufficiente aggiungere in **/etc/pam.d/login** la seguente linea:

```
account      required      pam_access.so
```

Impostando il flag di controllo a **required**, se il tentativo d'accesso non dovesse rispettare le regole definite in **/etc/security/access.conf** la richiesta fallira', dopo avere elaborato la parte restante dello stack di autenticazione.

Poiche' tutti gli utenti, ad esclusione di **root** e degli account di sistema appartengono ad uno tra i gruppi **users**, **administrators** o **sys_operators** e' stato scelto di negare l'accesso su qualunque tty a chiunque non appartenga a tale gruppo, inserendo la seguente riga in **access.conf**:

```
-: ALL EXCEPT administrators sys_operators users:ALL
```

Il file **/etc/security/access.conf**, una volta effettuata l'autenticazione viene letto dal modulo **pam_access**, che neghera' l'accesso diretto per tutti gli utenti che non appartengano ai gruppi elencati. Infatti, tendendo di eseguire un login come utente **root**, il sistema ha risposto con il messaggio **Permission denied**.

Questa configurazione, naturalmente, funzionera' correttamente solo nel caso in cui non esistano utenti di nome **administrators**, **sys_operators** o **users**, perche' in tal caso l'accesso verrebbe consentito a tali utenti, e non agli appartenenti ai gruppi omini.

Un'altra limitazione d'accesso che ho ritenuto utile era invece relativa al servizio **ssh**. Sull'host **paranoid** e' in esecuzione il servizio **ssh**, in modo che l'utente **rfc** possa connettersi da remoto dall'host **stargazer** ed eseguire alcune operazioni per il controllo dell'integrita' del filesystem (vedere sezione relativa ad **aide**). L'accesso **ssh** da parte di qualsiasi altro utente e' da ritenersi non autorizzato, quindi ho voluto fare in modo che **rfc** fosse l'unico abilitato a farlo.

Per tale motivo ho aggiunto la seguente riga in **/etc/pam.d/ssh**:

```
account      required      pam_access.so
accessfile=/etc/security/ssh_access.conf
```

Come si puo' notare e' stata utilizzata l'opzione **accessfile**, per specificare un altro file anziche' **/etc/access.conf**, dove sono definite le regole per gli utenti abilitati ad accedere al servizio **ssh**.

Avendo la necessita' di fare accedere il solo utente **rfc**, il file **ssh_access.conf** e' piuttosto semplice ed e'

costituito dalla riga:

```
-:ALL EXCEPT rfc rbackup:ALL
```

Secondo la documentazione sulla sintassi del file **access.conf**, nell'ultimo campo, anziche' **ALL** si sarebbe potuto utilizzare il nome dell'host da cui consentire l'account (nel mio caso **stargazer**) in modo da consentire l'accesso esclusivamente agli utenti **rfc@stargazer** e **rbackup@stargazer**.

Ho pero' verificato che con una simile configurazione qualsiasi utente da **stargazer**, era in grado di eseguire login via **ssh** su **paranoid**.

Con il file **ssh_access.conf** riportato, invece, l'accesso sarebbe consentito esclusivamente agli utenti **rfc** e **rbackup**, ma da qualsiasi host di provenienza. Questa configurazione da sola sarebbe forse meno sicura anziche' un **-:ALL EXCEPT rfc rbackup:stargazer**, ma abbinata al tcpwrapper ed al file **/etc/hosts.allow**, consente l'accesso ad **rfc** e **rbackup** dal solo host **stargazer**.

La necessita' di consentire accesso **ssh** all'utente **rbackup** verra' illustrata successivamente, nella sezione relativa al backup.

Sono consapevole che questa ulteriore misura di sicurezza potrebbe forse risultare ridondante rispetto alle altre operazioni gia' menzionate relative al servizio **ssh**, ma ho preferito ricorrervi comunque, poiche' potrebbe costituire comunque un limite per l'attacker, nel caso un sistemista dovesse sbagliare la configurazione del firewall, del servizio **ssh** stesso o del tcpwrapper.

- Limitare l'accesso in base all'orario: volendo, sempre tramite l'utilizzo di **PAM** e' possibile specificare delle limitazioni all'accesso degli utenti, in base a delle fasce orarie. Nel mio caso si potrebbe pensare di consentire l'accesso **ssh** all'utente **rfc** solamente per 15 minuti a partire dalle 3:00 am, poiche' l'operazione di "download" del database con le informazioni sullo stato dei files da **paranoid** a **stargazer** e' inserita all'interno di un cronjob eseguito appunto alle 3:00 am.

Per abilitare questo ulteriore controllo, e' necessario ricorrere al modulo **pam_time**, aggiungendolo nel modo seguente all'interno del file **/etc/pam.d/ssh**:

```
account      required    pam_time.so
```

Una volta effettuata l'autenticazione, quindi, il modulo **pam_time** andra' a leggere il file **/etc/security/time.conf**, ed a seconda del suo contenuto consentira' o meno l'accesso al servizio.

Ho quindi modificato il file in questione inserendo la linea:

```
ssh;*:rfc;A10300-0315
```

Il significato di tale linea e' quello di consentire l'accesso al servizio **ssh** all'utente **rfc** su qualsiasi terminale virtuale, qualsiasi giorno della settimana(**A1**) dalle 3:00 am alle 3:15 (**0300-0315**).

Per testare il funzionamento della regola, ho tentato di connettermi fuori dall'orario previsto, ma giustamente, il sistema ha risposto con il messaggio **Permission denied**.

Adottando questa soluzione pero', sara' necessario ricordarsi di disabilitare il controllo dell'orario (ad esempio commentando la riga in **time.conf** su **paranoid**) ogni volta che si eseguirà su **stargazer** il comando **remotefc.sh update paranoid**, a meno che non lo stia eseguendo nell'arco di tempo tra le 3:00 e le 3:15 am.

Personalmente ritengo questo controllo in base all'orario fin troppo paranoico, ma per completezza mi e' sembrato giusto provarlo e citare anche questa possibilita'.

- Limitare l'utilizzo del comando **su**. Come gia' illustrato in precedenza **su** consente a chi lo invoca di diventare un

altro utente, cambiando i propri **ID** e **GID**. Per default questo comando e' impostato con il bit **SUID** attivo e permesso di esecuzione per tutti gli utenti. In questo modo chiunque, conoscendo le rispettive password, e' in grado di "impersonare" un altro utente, persino **root**.

Un primo metodo molto brutale, potrebbe essere quello di togliere il bit **SUID** a **/bin/su**, in modo che nessuno, ad eccezione di **root**, possa usare tale comando. E' certamente una soluzione possibile, ma impedirebbe ad esempio di vietare i login diretti all'utente **root**, perche' poi non sarebbe piu' possibile cambiare **ID** tramite **su**. D'altro canto e' innegabile il fatto che i permessi di default associati al comando siano troppo generosi.

Una buona alternativa a questi estremi potrebbe essere quella di consentire l'utilizzo del comando su esclusivamente ad un insieme ristretto di utenti selezionati.

Un possibile scenario potrebbe essere quello in cui esistano due gruppi **sys_operators** e **administrators**. Agli utenti appartenenti al primo gruppo deve essere concesso l'utilizzo del comando **su**, ma senza pero' la possibilita' di diventare **root**, a differenza degli appartenenti ad **administrators**.

Naturalmente, una scelta del genere puo' risultare sensata nel caso ci sia la necessita' di avere piu' amministratori di sistema per la macchina in questione. Nella macchina usata per il progetto, la soluzione che andro' ad esporre, con questa divisione degli utenti tra gruppi e' chiaramente inutile, poiche' sono io l'unica persona a gestirla, ma e' stata fatta a titolo esemplificativo.

Per impedire l'utilizzo a chiunque del comando **su**, dunque sono stati rimossi i permessi di lettura ed esecuzione a tutti gli utenti con **chmod 750 /bin/su**, impostato il bit **SUID** per il proprietario del file con **chmod u+s /bin/su**, ed infine cambiato il gruppo associato al file con **chgrp sys_operators /bin/su**.

In questo modo, i semplici utenti non avranno i privilegi per eseguire tale comando, mentre i membri di **sys_operators** potranno utilizzare **su**, potendo pero' impersonare anche **root**.

Avendo pensato al gruppo **sys_operators** solo per utenti deputati ad eseguire piccoli interventi sulla macchina senza richiedere l'ausilio dell'amministratore di sistema, anche questa soluzione sembrerebbe troppo generosa nei loro confronti.

Dunque, per fare in modo che solo gli utenti del gruppo **administrators** possano utilizzare **su** per diventare **root**, e' stato fatto ricorso al modulo **pam_wheel**. Esso consente infatti di specificare uno specifico gruppo (per default il gruppo **wheel**) ai cui membri sia consentito di assumere **ID 0**.

A tale scopo e' stato modificato il file **/etc/pam.d/su** aggiungendo la linea :

```
auth required pam_wheel.so group=administrators
```

In questo modo se un utente del gruppo **sys_operators** dovesse cercare di eseguire **su** per diventare **root** il sistema risponderebbe con il messaggio **su: permission denied**, anche nel caso la password inserita per l'utente **root** fosse risultata corretta.

Provando ad eseguire il comando **su** con l'utente **davide** del gruppo **sys_operators** il comportamento mostrato dal sistema e' appunto stato il precedente. Provando ad impersonare un altro utente il comando ha invece funzionato correttamente.

Invocando invece **su** con l'utente **andrea**, appartenente al gruppo **administrators**, la procedura e' stata eseguita con successo, ritrovandosi ad accedere come utente **root**.

Infine, per via dei permessi impostati a **/bin/su**, si ricorda come sia fondamentale che gli utenti del gruppo **administrators**, vengano aggiunti anche a **sys_operators**, poiche' in caso contrario non avrebbero il permesso di esecuzione per tale comando.

Sistema di logging

Registrare le operazioni svolte sul server e' un'attivita' di importanza fondamentale, sia come fonte di informazioni utili per risolvere eventuali malfunzionamenti, che come forma di controllo del sistema, verificando che esso non stia subendo attacchi o tentativi di intrusione.

Un attacker abile certamente sara' in grado di occultare al meglio le proprie tracce, ad esempio cancellando i segni del propria operato dai file dove il log server registra tutte le attivita'. Emerge quindi la necessita' di proteggere questi file nel migliore modo possibile, e quindi non memorizzandoli esclusivamente sulla macchina che si desidera monitorare: in caso di violazione della stessa essi sarebbero a piena disposizione dell'attacker.

Una prima possibilita', che personalmente ritengo non sia troppo pratica, potrebbe essere quella di mantenere anche una copia cartacea dei log stessi. Certamente l'attacker, a meno di non avervi accesso fisicamente non potrebbe cancellare le sue tracce dalle stampe dei log. Questa scelta comporterebbe pero' un incremento mostruoso delle scartoffie che gia' tendono ad esistere in un ambiente di lavoro, oltre che problemi pratici per la loro analisi ed archiviazione.

Scartando questa ipotesi, si potrebbe pensare all'utilizzo di un secondo log server, in modo da replicare anche in remoto i log che vengono memorizzati localmente. In un simile caso l'attacker dovrebbe cancellare le sue tracce in due posti diversi. Inoltre, presupponendo che il log server remoto preveda tutti i possibili accorgimenti ai fini della sicurezza del sistema, tra cui l'installazione solo del software strettamente necessario, non sarebbe cosi' semplice per l'attacker cancellare le proprie tracce anche da esso.

Si presume inoltre che tale log server non sia direttamente connesso ad internet, ma sia accessibile esclusivamente dalla rete interna, minimizzando quindi la sua "esposizione".

La prima scelta da considerare e' quella del syslog-server. I piu' diffusi attualmente sono **syslog**, **metalog** e **syslog-ng**. **Syslog** e' il logger standard dei sistemi unix, ma ha la pesante limitazione di potere restare in ascolto esclusivamente su porte **UDP**. Non essendo un protocollo orientato alla connessione potrebbero andare persi alcuni pacchetti contenenti messaggi, ed inoltre, sarebbe decisamente piu' semplice per un'attacker inviare finti messaggi al log server essendo tali messaggi incapsulati all'interno di pacchetti **UDP** anziche' **TCP**.

Alla fine la mia scelta e' caduta su **syslog-ng**, sia per l'ottima granularita' nella definizione delle regole di filtraggio, che per la disponibilita' di maggiore documentazione. E' invece stato scartato **metalog**, poiche' non consente la registrazione dei log su host remoti.

Al solito la macchina da monitorare e' l'host **paranoid**, mentre il log server remoto sara' **stargazer**.

Su entrambe le macchine e' stato installato **syslog-ng** con il comando **apt-get install syslog-ng**. In questo modo **apt** ha provveduto anche alla rimozione di **syslog**, che era stato installato di default nel sistema.

Sull'host **paranoid** e' stato modificato il file **/etc/syslog-ng/syslog-ng.conf**, nel seguente modo:

- **sources** sono state definite tutte le "sorgenti" dalle quali raccogliere i dati da registrare nei log. Una sorgente e' un driver che non fa altro che raccogliere i messaggi usando un metodo predefinito. Nel mio caso sono state definite le seguenti sorgenti per l'host paranoid:

```
source s_all
{
    # messaggi generati da Syslog-NG
    internal();
    #standard Linux log source
    unix-stream("/dev/log" max-connections(10));
    # messaggi dal kernel
    file("/proc/kmsg" log_prefix("kernel: "));
};
```

Come si puo' notare, la prima sorgente **internal()** indica i messaggi generati dallo stesso **syslog-ng**, in modo da potere registrare anche avvertimenti ed errori relativi al log-server, come ad esempio il suo arresto o riavvio.

Unix-stream apre invece un socket unix sul quale ascolta i messaggi generati dal sistema. Per evitare attacchi di tipo DoS viene imposto anche un limite al massimo numero di connessioni accettate con l'opzione **max-connections**. Il valore 10 e' piuttosto ristretto, ma comunque adatto alla mia situazione, trattandosi di un server con poco carico di lavoro. Nel caso di sistemi con maggiore occupazione sara' opportuno aumentare questo valore.

L'ultimo driver **file()** infine legge i messaggi che il kernel scrive in **/proc/kmesg**.

- **destination**: le destinazioni costituiscono i punti dove inviare e memorizzare i messaggi raccolti dalle **sources** se le regole di filtraggio vengono soddisfatte. Come per le sorgenti esistono diversi driver che definiscono come distribuire tali messaggi.

```
# destinazioni standard
destination df_auth { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/auth.log"); };
destination df_syslog { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/syslog"); };
destination df_cron { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/cron.log"); };
destination df_daemon { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/daemon.log"); };
destination df_kern { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/kern.log"); };
destination df_lpr { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/lpr.log"); };
destination df_mail { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/mail.log"); };
destination df_user { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/user.log"); };
destination df_uucp { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/uucp.log"); };
destination df_sshd { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/sshd.log"); };

# destinazioni per le varie facility e livelli severity.
# Usate solo alcune per mail.info, mail.crit e mail.warn
destination df_facility_dot_info { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/$FACILITY.info"); };
destination df_facility_dot_notice { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/$FACILITY.notice"); };
destination df_facility_dot_warn { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/$FACILITY.warn"); };
destination df_facility_dot_err { file
  ("/var/log/hostnames/$HOST/$YEAR/$MONTH/$FACILITY.err"); };
destination df_facility_dot_crit { file
```



```
( "/var/log/hostnames/$HOST/$YEAR/$MONTH/$FACILITY.crit"); };  
[...]
```

#debug e messaggi vari

```
destination df_debug { file  
( "/var/log/hostnames/$HOST/$YEAR/$MONTH/debug"); };  
destination df_messages { file  
( "/var/log/hostnames/$HOST/$YEAR/$MONTH/messages"); };
```

#file con messaggi critici, situazione di emergenza!!!!

```
destination df_emergency { file  
( "/var/log/hostnames/$HOST/$YEAR/$MONTH/log.emerg"); };
```

#stampa su consoles virtuali

```
destination du_all { usertty("*"); };
```

#logging remoto su stargazer

```
destination stargazer { tcp("10.0.1.2" port(15514)); };
```

Come si puo' notare dal precedente stralcio del file di configurazione vengono definiti diversi file ove memorizzare i messaggi, a seconda della loro categoria, per esempio il file **auth.log**. E' infatti possibile definire dei filtri sui messaggi, ed in base ad essi scegliere dove memorizzarli. Da qui l'utilizzo di destinazioni diverse, in modo da favorire la leggibilita' e l'analisi dei log stessi.

Quasi tutte le destinazioni usano il driver **file()**, e quindi provvedono a memorizzare i messaggi che rispetteranno determinati criteri all'interno del file indicato come argomento.

Una particolarita' che si puo' notare e' l'utilizzo di macro quali **\$HOST**, **\$YEAR** e **\$MONTH**. Esse verranno rispettivamente espanse nell'hostname della macchina che ha generato il messaggio, l'anno ed il mese in cui esso e' stato ricevuto. Per default le macro **\$YEAR**, **\$MONTH** ecc. verrebbero espanse con l'anno e il mese in cui il messaggio e' stato generato, ma tale comportamento e' stato modificato dall'opzione **use_time_recvd()** come si vedra' successivamente. L'utilita' di questa opzione risultera' piu' chiara quando verra' trattata la configurazione del logserver remoto.

Ricorrendo a queste macro e' quindi possibile organizzare automaticamente i log registrandoli in cartelle a seconda della loro provenienza e di quando siano stati generati. Ad esempio i log del mese di dicembre 2005 generati da **paranoid** saranno memorizzati sotto la directory **/var/log/hostnames/paranoid/2005/12**.

Ritengo che una simile scelta sia giustificata dall'organizzazione piu' chiara che si e' in grado di ottenere, e dalla maggiore semplicita' nel predisporre successivi script di shell che provvedano a ruotare i log in base alla data.

L'ultima destinazione, a cui e' stato associato l'alias **stargazer** utilizza il driver **tcp()**, e nel caso riportato consente di inviare i messaggi syslog alla porta **tcp 15514** dell'indirizzo **IP 10.0.1.2** ovvero l'indirizzo dell'host **stargazer**. Naturalmente, perche' questi messaggi possano essere ricevuti il log-server remoto all'indirizzo **10.0.1.2**, dovra' avere tale porta tra i suoi sorgenti per la raccolta dei messaggi.

Come si puo' notare l'indirizzo associato a **stargazer**, in questo caso non e' come al solito **192.168.1.2**, poiche' viene fatto uso di una **vpn** per cifrare la connessione tra le due macchine, come verra' illustrato successivamente,

- **filter**: specificano come **syslog-ng** debba eseguire l'instradamento dei diversi messaggi. In pratica essi sono costituiti da un'espressione booleana che verra' valutata per decidere se e dove un messaggio dovra' essere registrato. Ai filtri, come per destinazioni e sorgenti e' possibile associare dei nomi, in modo che essi possano venire utilizzati anche in diversi log statements.

I filtri sono stati impostati secondo i valori di facility e severity indicate nella **rfc 3164** (The BSD syslog protocol). Per la precisione i filtri definiti in base alle facility sono stati i seguenti:

```
#messaggi con facility 4 o 10 (autenticazione e sicurezza.)
filter f_auth { facility(auth, authpriv); };
#messaggi con facility diversa da auth ed authpriv.
filter f_syslog { not facility(auth, authpriv, mail, news); };
#messaggi con facility 9, relativi al demone cron
filter f_cron { facility(cron); };
#messaggi relativi agli altri demoni di sistema (facility 3)
filter f_daemon { facility(daemon); };
#messaggi dal kernel (0)
filter f_kern { facility(kern); };
#messaggi relativi al sistema di stampa (facility 6)
filter f_lpr { facility(lpr); };
#messaggi relativi a mail (facility 2 )
filter f_mail { facility(mail); };
#messaggi relativi allo user-space (facility 1)
filter f_user { facility(user); };
#sottosistema uucp
filter f_uucp { facility(uucp); };
#attivit  server ssh
filter f_sshd { facility(local0); };
```

Una seconda serie di filtri, sono stati invece definiti a seconda del livello di severity dei messaggi. Ad ogni livello di severity corrisponde infatti un numero i cui significati sono rispettivamente messaggi di **debug** (livello 7), **info** (informazioni, livello 6), **notice** (informazioni significative, 5), **warning** (avvisi, 4), **error** (situazioni di errore, 3), **critical** (eventi critici, 2), **alert** (condizioni che richiedono intervento immediato, 1) ed infine **emergency** (sistema inutilizzabile, livello 0).

Tra i filtri definiti quindi :

```
#messaggi con severity < 6
filter f_at_least_info { level(info..emerg); };
#severity <5
filter f_at_least_notice { level(notice..emerg); };
#severity < 4
filter f_at_least_warn { level(warn..emerg); };
#severity <3
filter f_at_least_err { level(err..emerg); };
#severity < 2
filter f_at_least_crit { level(crit..emerg); };

#messaggi con severity debug e facility diversa da auth, authpriv,
# news e mail
filter f_debug { level(debug) and not facility(auth, authpriv,
news, mail); };
```

```

# messaggi con severity da info a error, facility diversa da auth,
# authpriv,cron, daemon, news e mail
filter f_messages {
    level(info..error) and not facility
    (auth,authpriv,cron,daemon,mail,news); };

#messaggi con severity critica
filter f_critical{ level(crit); };

# messages con severity emergenza!!!
filter f_emerg { level(alert, emerg); };

```

Come si potrà notare alcuni filtri sono definiti per completezza o perché inclusi nella configurazione di default, ma senza essere effettivamente utilizzati, come ad esempio tutti gli **f_at_least_***. Per completezza si è preferito lasciarli definiti, nel caso dovessero tornare utili in seguito. Inoltre, non venendo inseriti in alcun log statement non dovrebbero avere effetti negativi relativamente alle prestazioni del log server.

- **log:** log path o log statements, definiscono come collegare tra loro sorgenti destinazioni e filtri. In pratica uno statement specifica la destinazione (**destination**) per i messaggi provenienti dalle sorgenti (**source**) specificate e che rispettino il filtro **filter**.

Tutti i log statements vengono processati nell'ordine in cui sono definiti all'interno di **syslog-ng.conf**, e quindi un messaggio potrà anche essere inviato a destinazioni diverse se rispetta i rispettivi statements.

```

#registra messaggi con facility auth e authpriv in auth.log
log { source(s_all); filter(f_auth); destination(df_auth); };
#registra il messaggi che rispettano f_syslog in df_syslog
log { source(s_all); filter(f_syslog); destination(df_syslog); };
#registra messaggi relativi alla facility 9 in cron.log
log { source(s_all); filter(f_cron); destination(df_cron); };
#registra messaggi con facility 3 in daemon.log
log { source(s_all); filter(f_daemon); destination(df_daemon); };
#registra i messaggi del kernel in kern.log
log { source(s_all); filter(f_kern); destination(df_kern); };
# registra i messaggi del sistema di stampa in lpr.log
log { source(s_all); filter(f_lpr); destination(df_lpr); };
#messaggi con facility mail in mail.log
log { source(s_all); filter(f_mail); destination(df_mail); };
#messaggi relativi allo user-space in user.log
log { source(s_all); filter(f_user); destination(df_user); };
#messaggi dal server sshd
log { source(s_all); filter(f_sshd); destination(df_sshd); };
# messaggi con facility mail e severity pari almeno ad info in
# mail.info
log { source(s_all); filter(f_mail); filter(f_at_least_info);
    destination(df_facility_dot_info); };
# messaggi con facility mail e severity pari almeno a warning in
# mail.warn

```

```

log { source(s_all); filter(f_mail); filter(f_at_least_warn);
destination(df_facility_dot_warn); };
# messaggi con facility mail e severity pari almeno a errori in
# mail.err
log { source(s_all); filter(f_mail); filter(f_at_least_err);
destination(df_facility_dot_err); };
# messaggi severity debug e facility diversa da auth, authpriv,
# news e mail, vanno in debug
log { source(s_all); filter(f_debug); destination(df_debug); };
# severity tra info e error e facility diversa da
# auth,authpriv,cron,daemon,mail,news registrati nel file
# messages
log { source(s_all); filter(f_messages); destination
(df_messages); };
# messaggi con severity critica in log.emerg
log { source(s_all); filter(f_critical); destination
(df_emergency); };
# messaggi con severity emergency e alert in log.emerg e sulle
# tty
log { source(s_all); filter(f_emerg); destination(df_emergency);
destination(du_all); };

```

Come si puo' notare all'interno dei filtri e dei files di destinazione e' presente anche un filtro con facility pari a **local0**. Esso e' relativo all'attivita' del servizio **ssh**., che e' stato configurato in modo da generare messaggi con questo livello di facility personalizzato, in modo da registrarli in un file separato, **sshd.log** appunto.

- vengono infine definite una serie di opzioni, nella sezione **options** del file di configurazione. Tra le opzioni modificate rispetto alla configurazione di default:
 - **sync(3)** : indica il numero minimo di linee che debbono essere presenti nella coda di output dei messaggi, prima che vengano scritte in un file.
 - **group(sys_operators)** e **perm(0640)**: impone che i nuovi files creati da **syslog-ng** appartengano al gruppo **sys_operators** con permessi di lettura e scrittura per il proprietario e sola lettura per gli appartenenti al gruppo. Il proprietario dei files e' per default l'utente **root**, ma volendo e' possibile cambiarlo tramite l'opzione **owner**.
 - **dir_group(sys_operators)** e **dir_perm(0750)**: specifica che le nuove cartelle create da **syslog-ng** appartengano al gruppo **sys_operators**, ed abbiano permessi di lettura scrittura ed esecuzione per il proprietario (ovvero **root**), e di sola lettura ed esecuzione per gli appartenenti al gruppo. E' possibile cambiare l'owner delle directory tramite l'opzione **dir_owner**.
 - **check_hostname(yes)** : consente di eseguire una verifica sull'hostname dei messaggi ricevuti, se esso contiene caratteri validi o meno.
 - **keep_hostname(no)** : specifica se fidarsi o meno dell'hostname contenuto nei messaggi syslog. Impostandolo a no il valore di hostname dei messaggi verra' sempre riscritto a seconda della provenienza dei pacchetti che li contengono.
 - **use_time_recvd(yes)**: utilizzata per l'espansione delle macro, indica di sostituire le macro relative al tempo con il momento della ricezione del messaggio e non quello della spedizione del precedente log-server della catena.

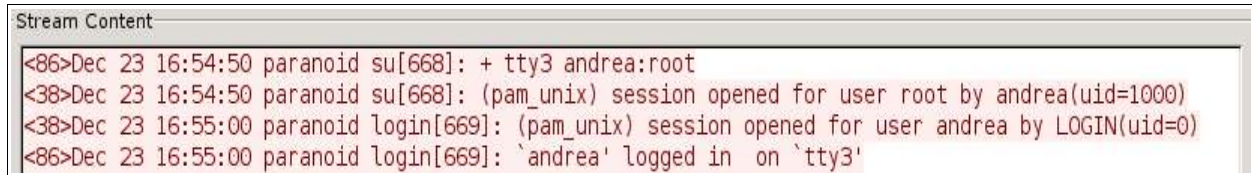
La configurazione di **syslog-ng** su **stargazer** e' pressoché identica a quella effettuata per **paranoid**. Le uniche differenze sono nella mancanza della destinazione `destination stargazer { tcp("10.0.1.2" port`

```
(15514)); }; e dell'aggiunta tra le sorgenti di tcp(ip("10.0.1.2" port(15514) keep-alive (yes)); .
```

In questo modo il log-server su **stargazer** registrerà anche i messaggi in arrivo sulla sua porta **tcp 15514**. Anche in questo caso l'indirizzo IP specificato non è **192.168.1.2**, ma **10.0.1.2**, ovvero quello dell'interfaccia virtuale usata per la **vpn**.

Un'altro aspetto considerato per il sistema di logging remoto è stata intatta la cifratura dei messaggi trasferiti. In caso contrario un eventuale attacker potrebbe avere la possibilità di osservare i messaggi inviati, sniffando il traffico sul segmento di rete che collega le due macchine.

Ad esempio dal seguente estratto di un capture l'attacker potrebbe venire a sapere che l'utente **andrea** è abilitato a diventare **root** tramite il comando **su**.



The screenshot shows a network capture window titled "Stream Content". It displays four lines of captured data in a monospaced font. The first line shows a successful login for user 'andrea' to 'root'. The second line shows a session opening for user 'root' by 'andrea'. The third line shows a session opening for user 'andrea' by 'LOGIN'. The fourth line shows 'andrea' logging in on 'tty3'.

```
<86>Dec 23 16:54:50 paranoid su[668]: + tty3 andrea:root
<38>Dec 23 16:54:50 paranoid su[668]: (pam_unix) session opened for user root by andrea(uid=1000)
<38>Dec 23 16:55:00 paranoid login[669]: (pam_unix) session opened for user andrea by LOGIN(uid=0)
<86>Dec 23 16:55:00 paranoid login[669]: `andrea' logged in on `tty3'
```

È quindi fondamentale cifrare il canale tra le due macchine. Le mie scelte iniziali per implementare questa soluzione sono state **stunnel** oppure **zebedee**. Quest'ultimo è stato scartato perché non essendo presente in nessuna versione di **debian**, neppure nella **testing**, non l'ho ritenuto sufficientemente testato per venire utilizzato su un server.

Riguardo a **stunnel**, mi è invece stato impossibile farlo funzionare. Secondo i log generati non riusciva ad accettare connessioni dall'esterno, riportando il codice di errore **111**. Facendo ricerche sia sui newsgroup che nella mailing list del progetto ho potuto constatare che il problema si è presentato anche ad altri, ma nessuno pareva essere riuscito a risolverlo. La documentazione, inoltre, non forniva alcuna informazione relativamente a tale errore.

Alla fine ho quindi deciso di ricorrere a **openvpn** per la creazione di una **vpn** tra **paranoid** e **stargazer**, anche se probabilmente si tratta di una soluzione sovradimensionata rispetto alle mie necessità.

Su entrambe le macchine è quindi stato installato **openvpn** con il comando **apt-get install openvpn**. Al solito **apt** ha provveduto anche alla creazione dei necessari script e link nel sistema di init, in modo da creare automaticamente la **vpn** all'avvio della macchina.

Per prima cosa è stata generata su **stargazer** la chiave da utilizzare con il comando **openvpn --genkey --secret vpn.key**, copiata all'interno di **/etc/openvpn**, ed assegnato il permesso di sola lettura all'utente **root** (**chmod 400 /etc/openvpn/vpn.key**).

Per evitare che essa potesse essere cancellata erroneamente le è stato associato anche l'attributo immutabile con **chattr +i /etc/openvpn/vpn.key**.

La chiave è poi stata copiata anche su **paranoid**, associandole gli stessi permessi.

Per fare funzionare la **vpn** è stato necessario l'utilizzo del dispositivo di rete virtuale tun/tap. Esso consente di creare un tunnel IP punto-punto a livello kernel. Ovviamente il kernel deve essere stato compilato includendo il supporto TUN/TAP. Non è stato necessario creare il device con il comando **mknod**, poiché aveva già provveduto a farlo **debconf** dopo l'installazione del pacchetto **openvpn**.

Il file di configurazione per **stargazer**, **/etc/openvpn/server.conf**, è stato il seguente :

```
#server vpn
dev tap
lport 5000
ifconfig 10.0.1.2 255.255.255.0
secret /etc/openvpn/vpn.key
verb 2
```

L'opzione **dev** indica il device utilizzato per il tunnel. I possibili device utilizzabili sarebbero tun e tap. La differenza dovrebbe stare nel fatto che tun si usa per la trasmissione di pacchetti IP e tap per i frame ethernet.

Lport specifica la porta usata lato server per la **vpn**, **ifconfig** l'indirizzo IP e la netmask dell'interfaccia virtuale, **secret** il percorso al file contenente la chiave, ed infine **verb** definisce il livello di verbosity.

Sull'host **paranoid** le operazioni eseguite sono state le medesime, a differenza di alcune differenze nel file di configurazione **/etc/openvpn/client.conf**:

```
#client vpn
remote 192.168.1.2
dev tap
rport 5000
ifconfig 10.0.1.4 255.255.255.0
secret /etc/openvpn/vpn.key
verb 1
```

Remote specifica il vero indirizzo IP della macchina a cui ci si andrà a connettere, ovvero **stargazer**, **rport** la porta utilizzata all'altra estremità per creare il tunnel, **secret** il file della chiave e **ifconfig** l'indirizzo dell'interfaccia virtuale su **paranoid**.

Infine, su **stargazer**, è stato necessario abilitare l'ip forwarding, scrivendo un uno nell'opportuno file sotto il filesystem **/proc** con il comando **echo 1 > /proc/sys/net/ipv4/ip_forward**. Per automatizzare l'operazione si potrebbe inserire il comando in qualche script eseguito automaticamente all'avvio del sistema.

Eseguite queste operazioni si è avviata la **vpn** su entrambe le macchine con la linea di comando **/etc/init.d/openvpn start**.

Configurando in questo modo la **vpn**, un eventuale attacker che dovesse sniffare il traffico tra le due macchine vedrebbe solo dei pacchetti **UDP** con porta sorgente e destinazione pari a **5000**, il cui contenuto sarebbe per lui privo di significato a causa della cifratura.

Per verificare il corretto funzionamento è stato controllato con **ethereal** lo scambio dei messaggi syslog tra **paranoid** e **stargazer** durante il login di un utente :

Source	Destination	Protocol	Info
192.168.1.2	192.168.1.4	UDP	Source port: 5000 Destination port: openvpn
192.168.1.4	192.168.1.2	UDP	Source port: openvpn Destination port: 5000
192.168.1.2	192.168.1.4	UDP	Source port: 5000 Destination port: openvpn
paranoid.local	EdimaxTe_41:08:cb	ARP	Who has 192.168.1.2? Tell 192.168.1.4
stargazer.local	paranoid.localdomain	ARP	192.168.1.2 is at 00:50:fc:41:08:cb
192.168.1.4	192.168.1.2	UDP	Source port: openvpn Destination port: 5000
192.168.1.2	192.168.1.4	UDP	Source port: 5000 Destination port: openvpn
73.e0.95	02.b7.42	FC	[0.24.14 <-- 0.55.13] Unknown frame
192.168.1.2	192.168.1.4	UDP	Source port: 5000 Destination port: openvpn
192.168.1.4	192.168.1.2	UDP	Source port: openvpn Destination port: 5000
192.168.1.2	192.168.1.4	UDP	Source port: 5000 Destination port: openvpn
192.168.1.4	192.168.1.2	UDP	Source port: openvpn Destination port: 5000
192.168.1.2	192.168.1.4	UDP	Source port: 5000 Destination port: openvpn

Come previsto il traffico visibile è costituito dai pacchetti **UDP** della **vpn**, mentre su **stargazer** il login è stato correttamente registrato nel file **/var/log/hostnames/paranoid/2005/12/auth.log** :

```
Dec 28 11:02:02 paranoid login[456]: (pam_unix) session opened for user
```

```
andrea by LOGIN(uid=0)
Dec 28 11:02:02 paranoid login[456]: `andrea' logged in on `tty2'
```

Si sottolinea inoltre, come sia scelto di non ricorrere ad un superdemone quale **inetd** per l'avvio e la gestione di **syslog-ng**, essenzialmente per motivi di performance, poiche' tramite il superdemone il servizio verrebbe avviato e gestito a seconda delle richieste giunte a **stargazer**.

Una simile soluzione sarebbe indicata per servizi che vengano utilizzati saltuariamente. Supponendo pero' che il servizio **syslog-ng** debba essere acceduto piuttosto frequentemente da **paranod**, si e' scelto di eseguirlo in modalita' stand-alone, ovvero avviandolo con il rispettivo script in **/etc/init.d**.

Se si fosse optato per l'utilizzo di un superdemone, esso avrebbe dovuto ogni volta reperire risorse necessarie da allocare per il processo da fare partire (nel caso **syslog-ng**). Il "demone figlio" avrebbe poi a sua volta dovuto essere caricato in memoria, leggere i propri file di configurazione, accettare la richiesta e cosi' via. Per un servizio che debba essere acceduto regolarmente e con una certa frequenza il dazio da pagare in termini di performances sarebbe stato troppo elevato.

Anche se memorizzati in maniera il piu' sicura possibile su un log-server remoto, l'utilita' di tali log sarebbe comunque estremamente ridotta se l'amministratore di sistema dovesse limitarsi ad ignorarli completamente.

Con ogni probabilita', in un ambiente di produzione un sistemista sara' oberato di lavoro e difficilmente avra' il tempo per tenere controllati accuratamente i log dei propri sistema. Uno strumento utile in casi del genere potrebbe essere **logcheck**. Certamente non puo' sostituire un'analisi attenta da parte di una persona preparata, ma e' comunque meglio di niente.

Si e' quindi provveduto ad installarlo su **stargazer** con il comando **apt-get install logcheck**.

Per la sua configurazione si e' intervenuti sui seguenti files nella directory **/etc/logcheck**:

- **logcheck.conf**
- **logcheck.logfiles**
- **header.txt**
- **/etc/cron.d/logcheck**

In **logcheck.conf** per prima cosa e' stato modificata l'opzione **DATE** nel modo seguente **DATE="\$(date+'%H:%M %d/%m/%Y')"**. Cosi' facendo la data riportate nei subject delle mail inviate all'amministratore sara' nel formato italiano anziche' americano, ovvero hh-mm gg/mm/yy.

E' stato impostato anche il livello di controllo da applicare ai messaggi, scegliendo tra workstation, server e paranoid. Si e' optato per il livello **server**.

In questo modo **logcheck** analizzera' tutte le stringhe che contengono le keywords riportate nei files sotto le directory **cracking.d** e **violation.d** escludendo quelle che rispettano le espressioni regolari definite sotto **ignore.d.paranoid** (**Nota**:il paranoid nel nome della directory non e' legato all'hostname della macchina usata per il progetto).

E' infine stata modificata l'opzione **SENDMAILTO**, per fare in modo di inviare le email di alert all'utente andrea@stargazer.localdomain anziche' **root**.

All'interno del file **logcheck.logfiles** sono invece stati elencati i files da monitorare. Nel mio caso, avendo diviso i log in sottodirectory, e' stato necessario riportare tra le entry all'interno di questo file, tutti i files **.log**, contenuti nelle directory relative ad anno e mese corrente per ogni host monitorato. Ad esempio per il mese di dicembre 2005 in **logcheck.logfiles** saranno elencati tutti i files che dovrebbero essere sotto **/var/log/hostnames/stargazer/2005/12/** e **/var/log/hostnames/paranoid/2005/12/**.

Per evitare l'incombenza della modifica mensile del file **logcheck.files** e' stato preparato il seguente shellscrip, **gen-logfiles.sh**, che lo faccia automaticamente alla mezzanotte del primo giorno di ogni mese.

```
#!/bin/bash
```

```

#estraggo anno e mese corrente
year=$(date +%Y)
month=$(date +%m)
#svuoto logcheck.logfiles del mese scorso
cat /dev/null > /etc/logcheck/logcheck.logfiles
#inserisco le nuove voci
for hostname in $(ls /var/log/hostnames/) ; do
echo /var/log/hostnames/$hostname/$year/$month/auth.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/syslog >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/cron.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/daemon.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/kern.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/lpr.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/mail.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/user.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/sshd.log >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/debug >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/messages >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/log.emerg >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/mail.crit >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/mail.info >> /
etc/logcheck/logcheck.logfiles
echo /var/log/hostnames/$hostname/$year/$month/mail.warn >> /
etc/logcheck/logcheck.logfiles
done

```

Per automatizzare tutte le operazioni di controllo dei log, sono state inserite le seguenti linee in /
etc/cron.d/logcheck.:

```

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
@reboot          logcheck    if [ -x /usr/sbin/logcheck ]; then nice
-n10 /usr/sbin/logcheck -R; fi
3 * * * *        logcheck    if [ -x /usr/sbin/logcheck ]; then nice
-n10 /usr/sbin/logcheck; fi

```



```
0 0 1 * * /etc/logcheck/gen-logfiles.sh
```

La prima linea imposta la variabile d'ambiente **PATH**, in modo che non ci siano problemi nell'invocare i vari programmi all'interno dello shellsript **/usr/sbin/logcheck**.

La seconda indica di eseguire **logcheck** ad ogni reboot della macchina, mentre la terza al terzo minuto di ogni ora.

Oltre ad eseguire **logcheck**, inoltre, tramite il comando **nice** viene alzato di 10 unita' il valore relativo alla priorita' del processo. Poiche' il range di valori ammissibile va infatti da -20 (priorita' massima) a 19 (priorita' minima), l'effetto del comando e' quello di assegnare una priorita' piu' bassa.

L'ultima linea consente di eseguire lo script **gen-logfiles.sh** alla mezzanotte del primo giorno del mese.

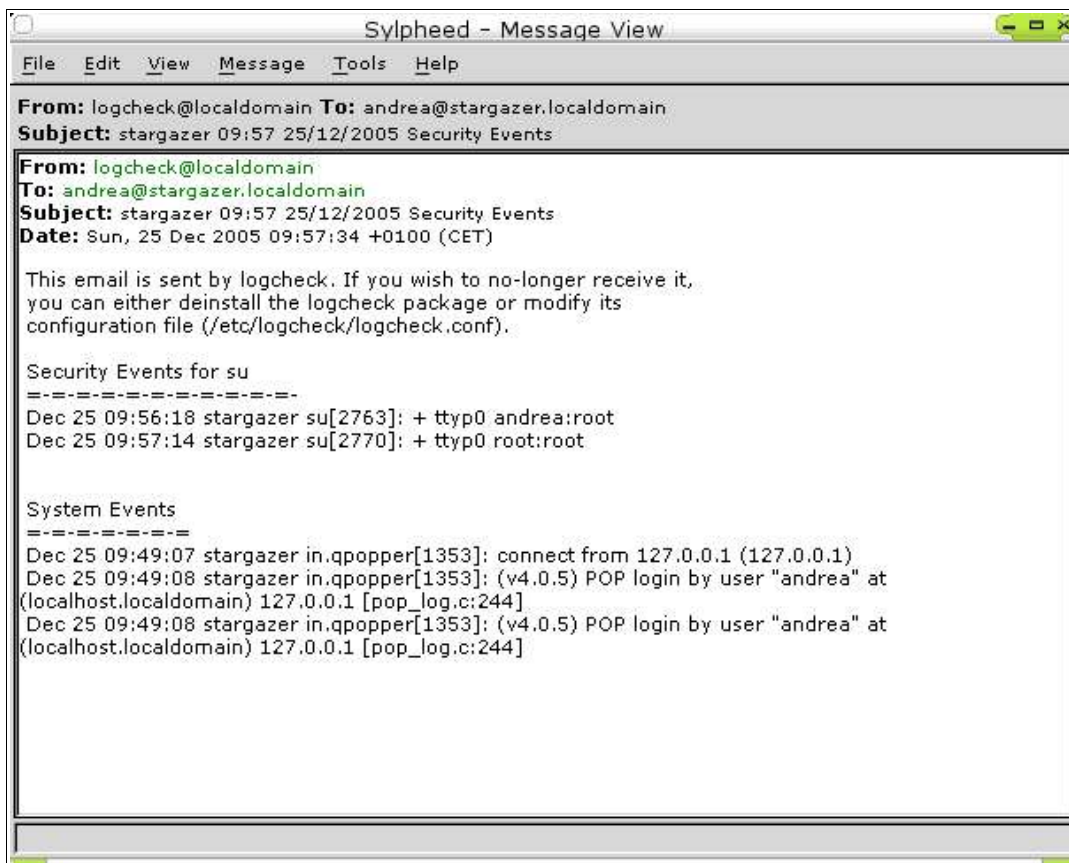
L'ultimo file di configurazione, **header.txt**, contiene l'intestazione delle mail di avviso inviate dopo i controlli da parte di **logcheck**. Si e' optato per lasciarla invariata:

```
This email is sent by logcheck. If you wish to no-longer receive it,
you can either deinstall the logcheck package or modify its
configuration file (/etc/logcheck/logcheck.conf).
```

Infine, perche' l'utente **logcheck** possa accedere ai files sotto la directory **/var/log/hostnames**, e' stato necessario aggiungerlo al gruppo **sys_operators**, con il comando **gpasswd -a logcheck sys_operators**.

Naturalmente, per non consentire il login al sistema all'utente **logcheck**, la sua password in **/etc/shadow** deve essere *, e la shell in **/etc/passwd** impostata a **/bin/false**.

Un'esempio di mail ricevuta e' la seguente :



L'ultima operazione per il sistema di logging e' stata la cosiddetta log rotation. E' infatti fondamentale gestire efficientemente la crescita in dimensione dei log di sistema, poiche' una volta saturato lo spazio disponibile per /**var/log**, anche la migliore architettura di logging risultera' completamente inutile. Questa operazione ha inoltre anche una certa importanza per mantenere ordine tra i log, ad esempio per archivarli su base settimanale o mensile, senza arrivare ad avere un unico file che contenga la storia di diversi mesi.

Tipicamente lo strumento usato per queste operazioni e' **logrotate**. Grazie ad esso e' possibile impostare dei limiti temporali o di spazio, in seguito ai quali i file di log vengono copiati e compressi, mentre l'originale viene sostituito da un file vuoto.

Nel mio caso ho optato per un'archiviazione mensile dei log, poiche' si tratta comunque solo di un piccolo server pensato a scopo didattico. Ovviamente in un ambiente di produzione, a seconda del traffico a cui esso sara' sottoposto, si potranno considerare altre politiche, come una rotazione settimanale ad esempio.

L'utilizzo di **logrotate** con i logfile organizzati nel mio modo, comporterebbe pero' dei piccoli problemi, poiche' ogni mese cambierebbe la directory entro cui essi si trovano. Una possibile soluzione potrebbe essere uno script simile **gen-logfiles.sh** usato per, che alla mezzanotte del primo giorno di ogni mese provveda a ricreare gli opportuni files di configurazione per **logrotate**.

Essendo pero' gia' i log divisi automaticamente in directory secondo l'host che li ha generati, il mese e l'anno di ricezione, ho preferito creare un altro shellscript, che provveda a comprimere il contenuto della cartella relativa al mese corrente.

Naturalmente, se la scelta prevista fosse quella di ruotare i log in base alla loro dimensione, potrebbe essere piu' opportuno ricorrere a **logrotate**, predisponendo uno script che all'occorrenza modifichi i suoi files di configurazione.

Lo script pensato per l'archiviazione dei log controlla la data corrente, estraendo con il comando **cut** il mese e l'anno. In base a tali valori e' possibile sapere la directory contenente i log relativi al mese precedente, e quindi per ogni host monitorato creare un file compresso con estensione tar.bz2 in **/var/log/rotate**.

In questo modo, all'interno di tale directory si troveranno i file compressi relativi ai log delle singole macchine, organizzati per mese. Ad esempio il file per il mese di novembre 2005 dell'host **paranoid** sara' **paranoid_2005_11.tar.gz**.

Per fare si' che le operazioni vengano eseguite per ogni host monitorato e' stato sufficiente un for sulle directory presenti in **/var/log/hostnames**, poiche' ognuna di essa corrisponde ad un diverso host che invia i propri syslog al log-server remoto.

Lo script consente inoltre di mantenere i log relativi agli ultimi mesi senza cancellarli, impostando il valore opportuno per la variabile **giacenza** all'interno dello script stesso.

Ad esempio se lo script fosse eseguito il primo di gennaio, e la variabile **giacenza** valesse 4, esso provvederebbe a cancellare le directory dei log relative al mese di Settembre. Perche' tali directory possano essere cancellate, devono pero' esistere in **/var/log/rotate** i relativi files **.tar.gz**. In caso contrario la directory verra' lasciata al suo posto, e verra' inviata un'email all'amministratore di sistema per avvisarlo della mancata cancellazione.

Lo script, **rotate-log.sh**, preparato per eseguire tali operazioni e' il seguente :

```
#!/bin/bash
#mesi di giacenza prima di cancellare
giacenza=2
#funzione che decrementa mese e anno se necessario
data_minus()
{
    if [ $month -eq 1 ]
    then
        month=12
```

```

        let year=$year-1
    else
        let month=$month-1
    fi
}

#calcolo data e anno
year=$(date +%Y)
month=$(date +%m)

#decremento e controllo
data_minus

#creo nuovo tarball per il mese precedente (nota hostname lcase)
for hostname in $(ls /var/log/hostnames/)
do
    if [ -d /var/log/hostnames/$hostname ]
    then
        tar -czvf /var/log/rotate/$hostname\_$_year\_$_month.tar.bz2
/var/log/hostnames/$hostname/$year/$month > /dev/null
    fi
done

#ricalcolo la data
year=$(date +%Y)
month=$(date +%m)

#cancello log di n mesi fa se esistono i relativi backup
for ((i=1; i<=$giacenza; i++))
do
    data_minus
done

for hostname in $(ls /var/log/hostnames/)
do
    if [ -f /var/log/rotate/$hostname\_$_year\_$_month.tar.gz ]
    then
        rm -rf /var/log/hostnames/$hostname/$year/$month
    else
        echo File /var/log/rotate/$hostname\_$_year\_$_month.tar.gz
mancante!!! Se lo si desidera eliminare manualmente la directory /
var/log/hostnames/$hostname/$year/$month | mail -s " Log Rotation
Error" andrea@stargazer.localdomain
    fi
done

```

Infine, per automatizzarne l'esecuzione mensile lo script e' stato copiato sotto la directory **/etc/cron.monthly** .

Naturalmente sara' a cura dell'amministratore provvedere a copiare su un altro dispositivo i file **tar.gz** creati dallo script ogni mese, impedendo che essi possano giungere all'esaurimento dello spazio a disposizione.

Backup

Un'ultimo aspetto da tenere in considerazione, prima di iniziare a trattare il kernel del sistema e' la politica di backup. E' scontata la necessita' di effettuare backup periodici dei dati, in modo da avere la possibilita' di recuperarli in caso di malfunzionamento.

Un primo elemento da definire sono i files dei quali si desidera eseguire il backup. Si potrebbero considerare due backup: di sistema e dei dati utente. I dati utente possono essere facilmente individuati per il fatto di essere solitamente mantenuti sotto la directory **/home**, mentre il sistema e le applicazioni utilizzano il resto del disco. Personalmente tra i dati utenti farei rientrare anche altri elementi quali la mailbox dell'utente, il proprio sito web (se la macchina che stiamo considerando e' anche un webserver) e cosi' via...

Se per un utente e' fondamentale recuperare i suoi dati, lo scopo del backup di sistema dovrebbe essere quello di minimizzare lo sforzo per riportarlo alla situazione precedente al verificarsi del problema. Nel secondo caso potrebbe quindi essere sufficiente eseguire un backup dei file di configurazione e di eventuali software che siano stati implementati "su misura". I restanti programmi possono infatti essere sempre recuperati dai cd d'installazione della distribuzione o da qualche repository tramite **apt**.

Per il backup di sistema ritengo quindi sia fondamentale salvare tutti i files di configurazione sotto **/etc** ed eventuali programmi che siano stati modificati, ad esempio a partire dai loro sorgenti, di cui non si abbia altra copia.

Alto elemento da salvare potrebbe essere la lista dei pacchetti installati, in modo da sapere esattamente cosa reinstallare. Tale lista puo' essere generata con il comando **dpkg -l** e rediretta in un file di testo che si provvedera' a copiare.

La frequenza con cui eseguire i diversi backup e' un secondo aspetto da tenere in considerazione. Ad esempio i backup di sistema potrebbero essere fatti solo in seguito a modifiche della configurazione, aggiornamenti ed installazione di nuovi software. I backup dei dati utente dovranno invece essere piu' frequenti.

Personalmente, per i dati utente, ritengo potrebbe essere una soluzione accettabile eseguire un backup completo a settimana (ad esempio la domenica) ed un backup incrementale per i restanti giorni della settimana. In questo modo sara' possibile recuperare i dati partendo dall'ultimo backup completo, ed "integrandolo" con il contenuto degli eventuali backup incrementali effettuati successivamente, e prima dall'evento che ha reso necessario il restore.

Nel mio caso, ho scelto di applicare questa politica di backup sia per i dati utente che per il backup di sistema. Non avendo installato o modificato particolari software ho infatti la necessita' di copiare solamente la cartella **/etc**, che ha comunque dimensioni piuttosto ridotte.

Per la precisione ho scelto di eseguire il backup delle seguenti cartelle :

- **/home**
- **/root**
- **/etc**
- **/var**, escludendo la directory **/var/log/hostname/paranoid** perche' replicata tramite **syslog-ng** sul log-server remoto.

Ritengo inoltre opportuno eseguire una copia del kernel e del relativo file **.config** in seguito ad ogni ricompilazione dello stesso, in modo da poterlo ripristinare il prima possibile in caso di necessita'. Non e' stata fatta alcuna menzione alla directory **/lib/modules**, perche' come verra' illustrato nel capitolo successivo ho scelto di preparare un kernel senza il supporto per i moduli.

L'ultimo aspetto da considerare e' la scelta del mezzo di backup, che dovra' essere in funzione della frequenza dei backup, della criticita' dei dati, la loro mole e dal budget a disposizione.

Se non dovessero esistere vincoli economici la soluzione migliore potrebbe essere l'uso di una **SAN** (storage area network), sia per quanto riguarda la prestazioni che lo spazio a disposizione. Infatti, una soluzione simile avrebbe teoricamente una capacita' di espansione illimitata, poiche' basterebbe collegare piu' switch tra loro per aumentare lo

spazio disponibile.

Altre possibilita' e' l'uso di un unita' nastro, o di un dat array esterno. L'uso di nastri dat ha il grande vantaggio del costo ridotto dei supporti, e dell'elevata capacita' di memorizzazione. Esse pero' hanno una fallibilita' superiore alla media se non conservate con sufficiente cura, e sono particolarmente sensibili ai campi elettromagnetici. Parte di questi problemi sono risolvibili tramite l'uso di un dat array, ma esso ha lo svantaggio di un costo non indifferente, ed in molti modelli della necessita' dell'intervento di un tecnico per il recupero delle cassette dallo stesso.

Un'alternativa piu' semplice ed economica potrebbero essere l'utilizzo di un masterizzatore e cd o dvd come supporto per i backup. Si tratta pero' di una soluzione abbastanza scomoda, vista la ridotta capacita' di memorizzazione dei supporti, e quindi indicata solamente per backup di quantita' di dati ridotte.

Un'ultima possibilita' interessante e' l'utilizzo di un server centralizzato su cui memorizzare il backup. Nelle versioni piu' semplici potrebbe essere sufficiente l'utilizzo di un fileserver, o la memorizzazione tramite **rsync**. Esistono pero' soluzioni molto piu' evolute, pensate appositamente per il backup quali **amanda** oppure **bacula**. Una trattazione approfondita di tali software esula dall'argomento di questo progetto, poiche' non specificatamente incentrato sul backup. Inoltre una soluzione del genere sarebbe piu' indicata per ambienti di produzione, dove esista appunto la necessita' del backup via rete di piu' macchine. Nel mio caso sarebbe una soluzione troppo superiore alle mie esigenze...

Non avendo a disposizione una san od un unita' a nastri, ho optato per eseguire il backup localmente e copiarlo su una terza macchina tramite **scp**. La macchina di cui eseguire il backup e' al solito **paranoid**, mentre l'host su cui replicarli **aqualung**, il cui indirizzo IP e' **192.168.1.5**.

Come strumento per eseguire il backup ho invece opato per **tar**, scartando **dump** poiche' pensato esclusivamente per filesystem **ext2/ext3**. Essendo **ext3** il filesystem del mio server, **dump** avrebbe potuto essere una soluzione accettabile. Ho preferito pero' scegliere **tar** nel caso un giorno si dovesse optare per un cambio di filesystem, in modo da avere meno problemi per recuperare i backup fatti per i precedenti fs. Infatti, a differenza di **dump**, **tar** non legge i dati in modalita' raw, ovvero senza interpretare la struttura del filesystem.

Come per il controllo dell'integrita' del filesystem, si e' presentata la necessita' di eseguire il comando **scp** all'interno di uno shellscript, e si e' quindi optato per un meccanismo di autenticazione basato su chiave asimmetrica.

L'idea e' quella di creare degli archivi **.tar** sull'host **paranoid**, nell'home directory di un apposito utente aggiunto a tale scopo: **rbackup**.

Creato l'archivio nella home directory dell'utente lo script provvede a cambiarne i permessi consentendo lettura e scrittura al solo proprietario. Avendo impostato un opportuno valore di **umask** tale operazione non sarebbe necessaria, ma ho preferito eseguirla comunque, in modo che anche in caso di errata definizione del parametro i permessi venissero cambiati in modo opportuno.

Si sottolinea, inoltre, il fatto che sia stata cambiata la home directory dell'utente **rbackup**, editando opportunamente il file **/etc/passwd**. La scelta, e' stata fatta per potere memorizzare i backup su una partizione separata, diversa da quella usata da **/home**. In tal caso se si dovesse esaurire lo spazio nella partizione montata sotto backup, l'unico effetto sarebbe l'impossibilita' di effettuare le copie, ma non avrebbe altre conseguenze sul funzionamento dell'host **paranoid**.

In un secondo momento, una volta che su **paranoid** e' stato creato l'archivio con il backup, su **aqualung** verra' invocato un secondo shellscript che provvede a copiarlo tramite **scp**. Agendo in questo modo, resta una finestra di tempo tra l'esecuzione del primo script su **paranoid**, e del secondo su **aqualung**, in cui l'ultimo backup potrebbe essere manipolato da un eventuale attacker che abbia ottenuto il controllo di **paranoid**. Ritengo pero' che sia una soluzione migliore rispetto ad avviare la copia da **paranoid**, e consentire quindi l'accesso via **ssh** anche su **aqualung**. In questo modo l'eventuale attacker potrebbe anche arrivare a cancellare o modificare i backup replicati sull'host remoto.

Con la prima soluzione, si ha comunque una maggiore certezza del fatto che l'ultimo backup prima di un eventuale attacco non sia stato modificato, essendo memorizzato su **aqualung**, che naturalmente non presenta alcun servizio attivo, e si limita esclusivamente ad eseguire la copia dei backup dalle macchine remote.

Una volta eseguita la copia, sempre tramite lo stesso script eseguito su **aqualung**, sarà possibile cancellare dagli host remoti gli archivi appena copiati. Nel mio caso l'unico host di cui si esegue il backup è **paranoid**, ma il discorso potrebbe essere esteso anche a più macchine.

Per evitare che il trasferimento dei backup possa avere effetti negativi a causa dell'elevato traffico di rete che causerebbe, si potrebbe pensare di aggiungere una seconda scheda di rete ad entrambi gli host e collegarli direttamente tra loro, se possibile. Ovviamente la connessione ssh deve essere fatta verso l'indirizzo ip assegnato alla seconda scheda di rete.

Una possibile soluzione per minimizzare la finestra “di esposizione” dei backup su **paranoid**, potrebbe essere quella di creare un piccolo programma client-server che si limiti ad inviare al server di backup (nel caso **aqualung**) un pacchetto contenente un messaggio che avvisi del completamento della costruzione degli archivi. Ricevuto tale messaggio il server avvia lo script che esegue la copia degli archivi tramite **scp**. In questo progetto non ho comunque implementato questa soluzione, essenzialmente per motivi di tempo, ma ritengo che sia una possibilità da considerare.

Per consentire l'esecuzione del comando **scp** all'interno di uno script, senza alcuna richiesta di password, si è fatto ricorso ancora all'autenticazione tramite chiave asimmetrica come per il controllo remoto dell'integrità del filesystem. Anche sull'host **aqualung** si è quindi creato l'utente **rbackup**, assegnandogli come home directory la cartella **/backup**, montata su una partizione separata dal resto del sistema. E' poi stata generata la coppia di chiavi da utilizzare, copiata la chiave pubblica su **paranoid** ed inserita tra quelle autorizzate nel file **/backup/.ssh/authorized_keys2**.

Sulle implicazioni di questo tipo di autenticazione, valgono le stesse considerazioni fatte in precedenza trattando l'integrità del filesystem.

Le uniche modifiche relative alla configurazione del server ssh visto in precedenza sono l'aggiunta dell'utente **rbackup@192.168.1.5** tra quelli ammessi ad accedere al sistema, modificando la direttiva **AllowUsers** di **/etc/sshd_config**

```
AllowUsers rfc@192.168.1.2 rbackup@192.168.1.5
```

E' inoltre stato modificato anche il file **/etc/hosts.allow**, in modo che il TcpWrapper consentisse l'accesso ssh anche da **192.168.1.5**:

```
#hosts.allow
sshd: 192.168.1.2 192.168.1.5
```

Sempre per consentire l'accesso è stata aggiunta anche la seguente regola per **netfilter** tramite **iptables**:

```
iptables -A INPUT -p tcp --dport 22 -s 192.168.1.5 -j ACCEPT
```

I backup su **paranoid** vengono eseguiti tramite il seguente shellscript **backup_tar.sh** copiato sotto la directory **/sbin**, che invoca opportunamente il comando **tar**. Per la precisione viene eseguito un backup completo il primo giorno di ogni mese ed ogni domenica. Il backup mensile è permanente, mentre quello settimanale viene sovrascritto alla settimana seguente.

```
#!/bin/bash
# Based on a script by Daniel O'Callaghan <danny@freebsd.org>
# and modified by Gerhard Mourani <gmourani@videotron.ca>

#check che script avviato da root
```

```

if [ $UID != 0 ]; then
    echo ATTENZIONE: solo l'utente root puo' avviare lo script
    exit 1
fi

# hostname
COMPUTER=paranoid
# directory da copiare
DIRECTORIES="/home /root /etc /var"
#directory da escludere
EXCLUDE="/var/log/hostnames/paranoid"
# dove creare gli archivi
BACKUPDIR=/backup
# file con data ultimo backup completo
TIMEDIR=/backup/last-full
# percorso tar
TAR=/bin/tar
# giorno della settimana
DOW=$(date +%a)
# giorno del mese
DOM=$(date +%d)
# Data
DM=$(date +%d-%b-%y)

# Backup mensile completo
if [ $DOM = "01" ]; then
    $TAR --exclude=\'$EXCLUDE\' -cf $BACKUPDIR/$COMPUTER-$DM.tar
$DIRECTORIES
    chown rbackup:rbackup $BACKUPDIR/$COMPUTER-$DM.tar
    chmod 600 $BACKUPDIR/$COMPUTER-$DM.tar
fi

# Backup settimanale completo
if [ $DOW = "Sun" ]; then
    NOW=$(date +%d%b%Y)
    # Update full backup date
    echo $NOW > $TIMEDIR/$COMPUTER-full-date
    $TAR --exclude=\'$EXCLUDE\' -cf $BACKUPDIR/$COMPUTER-$DOW.tar
$DIRECTORIES
    chown rbackup:rbackup $BACKUPDIR/$COMPUTER-$DOW.tar
    chmod 600 $BACKUPDIR/$COMPUTER-$DOW.tar

# backup incrementali, sovrascrivendo archivi sett. precedente
else
    # estrae data ultimo backup completo
    NEWER="--newer $(cat $TIMEDIR/$COMPUTER-full-date)"
    $TAR --exclude=\'$EXCLUDE\' $NEWER -cf $BACKUPDIR/$COMPUTER-

```



```

$DOW.tar $DIRECTORIES
    chown rbackup:rbackup $BACKUPDIR/$COMPUTER-$DOW.tar
    chmod 600 $BACKUPDIR/$COMPUTER-$DOW.tar
fi

```

Perche' lo script possa essere eseguito correttamente la prima volta, e' necessario creare le directory **/backup** e **/backup/last-full**. In **/backup/last-full**, inoltre, deve essere inserito il file contenente la data dell'ultimo backup completo. Ad esempio se l'ultimo backup completo risale ad oggi sara' sufficiente il comando `date +%d%b%Y > /backup/last-full/paranoid-full-date`. Poiche' lo script esegue backup completi solo la domenica ed il primo giorno di ogni mese, se lo si dovesse eseguire per la prima volta in un un giorno infrasettimanale sara' necessario crearsi manualmente il primo backup completo con il comando `tar --exclude=/var/log/hostnames/paranoid -cf paranoid-gg-mm.tar /etc /home /var /root`.

Volendo sarebbe anche possibile comprimere i files creati, tramite le opzioni **-z** o **-j** del comando **tar**, ma ho preferito evitarlo. Chiaramente si sarebbe potuto risparmiare dello spazio, ma secondo la mia opinione il rischio di corrompere l'archivio durante la compressione non giustificherebbe tale risparmio.

Lo script verifica la data corrente ed il giorno della settimana tramite le condizioni `if [$DOW = "Sun"]` e `if [$DOM = "01"]`, ed in caso esso sia una domenica od il primo del mese crea un archivio nella directory indicata da **BACKUPDIR** con nome nel formato **hostname-giorno-mese-anno** nel caso del backup mensile, oppure **HOSTNAME-giornodellasettimana** nel caso di backup settimanali. Ad esempio il backup del mese di dicembre sara' **paranoid-01-Dec-05.tar**, mentre il backup incrementale del venerdi' **paranoid-Fri.tar**.

L'esecuzione dei backup incrementali e' resa possibile dall'opzione **--newer** del comando **tar**, che consente di archiviare solo i files modificati in seguito alla data registrata in **/backup/last-full/paranoid-full-date**. Naturalmente ad ogni backup completo, viene aggiornata la data in tale file con il comando `echo $NOW > $TIMEDIR/$COMPUTER-full-date`.

Dal backup vengono inoltre escluse le directory definite dalla variabile **EXCLUDE**, usandola come argomento dell'opzione **--exclude** del comando **tar**.

Per automatizzare la sua esecuzione esso e' stato inserito all'interno di un cronjob dell'utente **root**, poiche' e' necessario copiare anche file che in caso contrario non sarebbe possibile leggere. Tramite il comando **crontab -e** e' quindi stata aggiunta la seguente linea, per eseguirlo ogni giorno cinque minuti dopo la mezzanotte:

```

5 0 * * * /sbin/backup_tar.sh

```

Per impedire l'esecuzione dello script ad altri utenti diversi dall'utente **root** sono stati cambiati i permessi ad esso assegnati con `chmod 700 /sbin/backup_tar.sh`. Così facendo lo script sara' eseguibile esclusivamente tramite **sudo** dagli utenti abilitati in **/etc/sudoers**, o da coloro a cui e' consentito impersonare **root** tramite **su**.

Per eseguire la copia dei backup creati su **paranoid** verso l'host remoto **aqualung**, su quest'ultimo e' stato creato un altro shellscript: **scp_backup.sh**. Esso, in base alla data odierna esegue la copia dell'opportuno file **.tar** da **paranoid** ad **aqualung** tramite il comando **scp**. La struttura dello script e' simile a quella di **backup_tar.sh**, cambiano i comandi eseguiti in base alle diverse condizioni soddisfatte.

Inoltre, anziche' specificare direttamente il nome dell'host da cui prelevare il file, esso deve essere indicato come primo parametro all'invocazione dello script. In questo modo esso potra' essere utilizzato per recuperare i backup da diversi host, cambiando solo il parametro passatogli. Inoltre, perche' lo script possa funzionare, per ogni host "controllato" deve esistere la relativa cartella in **BACKUPDIR**. Nel mio caso si tratta di **/backup/paranoid**:

```

#!/bin/bash
# ===== scp_backup.sh =====

```

```

if [ -z $1 ]; then
    echo "ERRORE : hostname non specificato"
    echo "Usage: $0 hostname"
    exit 1
fi

# dove creare gli archivi
BACKUPDIR=/backup
# giorno della settimana
DOW=$(date +%a)
# giorno del mese
DOM=$(date +%d)
# Data
DM=$(date +%d-%b-%Y)

# copia backup mensile completo
if [ $DOM = "01" ]; then
    scp rbackup@$1:$BACKUPDIR/$1-$DM.tar $BACKUPDIR/$1 | mail -s
    "Download backup $1" andrea@stargazer.localdomain
else
    # backup giornalieri, sovrascrivendo archivi sett. precedente
    scp rbackup@$1:$BACKUPDIR/$1-$DOW.tar $BACKUPDIR/$1 | mail -s
    "Download backup $1" andrea@stargazer.localdomain
fi
exit 0

```

Eseguita la copia dell'archivio, infine, viene inviata un'email per avvisare del completamento dell'operazione. In questo modo l'amministratore di sistema, o chi per esso, in caso non dovesse ricevere la mail di conferma giornaliera potrà rendersi subito conto che qualcosa non ha funzionato a dovere ed intervenire di conseguenza.

Per automatizzare l'esecuzione giornaliera dello script, esso e' stato inserito all'interno di un cronjob per l'utente **rbackup**, con il comando **crontab -u rbackup -e** :

```

5 1 * * * /backup/scp_backup.sh paranoid

```

Nell'esempio si e' deciso di eseguire la copia alle ore 1:05 am, ma la scelta deve dipendere anche dalla quantita' di dati che andranno a costituire gli archivi .tar sugli host, per non correre il rischio di eseguire **scp_backup.sh** sul backup server prima che l'archivio sia stato completato sull'altra macchina.

Ovviamente, poiche' lo script e' eseguito con un cronjob dell'utente **rbackup**, tale utente deve essere tra quelli abilitati in **/etc/cron.allow** .

Inoltre, se si utilizza il modulo **pam_time.so** per gestire gli orari di accesso al servizio **ssh**, deve essere modificato opportunamente il file **/etc/security/time.conf** sull'host **paranoid**, in modo da comprendere tra gli intervalli di tempo in cui e' consentito l'accesso **ssh** quelli in cui **aqualung** dovra' eseguire lo script **scp_backup.sh** per scaricare i backup tramite **scp**.

Perche' questi script possano funzionare correttamente, infine, e' necessario che le diverse macchine siano tra loro sincronizzate. Si immagini ad esempio il caso in cui la data di **aqualung** sia mercoledi' 21 dicembre, e quella di

paranoid venerdi' 23 dicembre: quest'ultimo creara' un archivio chiamato **paranoid-Fri.tar**, mentre lo script eseguito sul backup server scaricherebbe **paranoid-Wed.tar**. Da qui la necessita' di mantenere le macchine sincronizzate tra loro.

Kernel

Nell'ultima parte di relazione vorrei coprire alcuni aspetti relativi alla ricompilazione del kernel, che ritengo particolarmente importanti ai fini della sicurezza del sistema.

Il kernel compilato e' stato un kernel vanilla della serie **2.4**, per la precisione l'ultima versione disponibile su kernel.org, ovvero la **2.4.32**. Se possibile avrei preferito utilizzare uno dei kernel, disponibili nei repository di **apt**, appositamente patchati per debian. Ho invece optato per un kernel vanilla perche' i kernel debian successivi al **2.4.20** non sono compatibili con la patch grsecurity, che desidero invece usare.

Ritengo che il compilarci un kernel su misura sia una necessita' fondamentale, soprattutto per il rispetto della "regola d'oro". I kernel precompilati gia' disponibili con le varie distribuzioni sono pensati per un utilizzo piu' generale e per funzionare con piu' hardware possibile. Essi includono quindi diverse parti di codice relative ad hardware che magari non si possiede, o features di cui non si ha bisogno. Da qui la necessita' di costruirsi un kernel che contenga solo cio' di cui si necessita veramente, escludendo tutto il superfluo.

Praticamente tutti i kernel precompilati per le varie distribuzioni, prevedono anche il supporto per i moduli, ovvero parti di codice che possono essere caricate e rimosse dinamicamente dall'utente **root**, con comandi quali **modprobe**, **insmod**, **rmmod** ecc.

Essi sono sicuramente una grandissima comodita' su un sistema desktop, poiche' consentono di aggiungere funzionalita' senza ricompilare l'intero kernel. Ad esempio, in caso di cambio di una vecchia scheda video con una piu' recente gforce, per ottenere il supporto al nuovo hardware sara' sufficiente compilare il modulo necessario ed installarlo con un **modprobe nvidia** e pochi altri comandi.

In un ambiente di produzione, pero', si presume che sia noto in anticipo quale debba essere la configurazione e l'hardware a disposizione. Le successive modifiche hardware, inoltre, dovrebbero essere piuttosto rare, quindi si potrebbe anche considerare l'intera ricompilazione del kernel in caso di necessita'.

I moduli garantiscono quindi una grande flessibilita', ma espongono pero' il fianco a delle problematiche piuttosto gravi dal punto di vista della sicurezza. Si pensi ad esempio ai root-kit.

Un classico rootkit che sostituisce i comandi di sistema come **who**, **netstat**, **ps**, **ls** con versioni modificate, ed installa dei tool per rimuovere le tracce dei log puo' essere facilmente scoperto dall'amministratore di sistema. Ad esempio l'output del comando **ps** mostrerebbe meno processi di quanti visualizzati in **/proc**, dove per ognuno di essi esiste la relativa directory, il cui nome corrisponde al **PID** del processo stesso.

In caso di dubbi inoltre, si potrebbe provare ad utilizzare dei binari sicuri, ad esempio presi dal cd della distribuzione, per verificare lo stato del sistema.

Esistono pero' anche rootkit molto piu' "subdoli", che sfruttano la capacita' del kernel di supportare i moduli, e detti appunto rootkit **LKM** (Loadable Kernel Modules), che andando ad integrarsi direttamente nel kernel, possono arrivare a modificarlo profondamente.

Rootkit di quest tipo potrebbero sostituire gli interrupt handler del kernel con una nuova funzione, modificando l'**interrupt descriptor table** (fonte: phrack magazine n. 59 : handling the interrupt descriptor table). Scopo di questa tabella e' quello di associare un interrupt handler ai diversi tipi di interrupt.

Quando viene catturato un interrupt, il controllo passa all'interrupt handler, il cui indirizzo e' appunto riportato nella **interrupt descriptor table**. I rootkit **LKM** tipicamente modificano le entry di questa tabella, in modo da puntare a delle nuove funzioni, definite dal rootkit stesso.

Un altro sistema, invece, potrebbe essere quello di modificare la funzione associata all'interrupt, aggiungendo un jump ad una propria funzione sostitutiva, all'inizio della syscall legittima. Attualmente non sono pero' conosciuti rootkit che sfruttino questo sistema.

I rootkit **LKM**, lavorando direttamente come moduli del kernel sono quindi in grado di intercettare le chiamate di sistema di qualsiasi processo e modificarne il risultato senza fare venire meno l'integrita' dei binari dei programmi. Esempio di funzionalita' svolte da tali rootkit sono:

- impostare la scheda di rete in modalita' promiscua
- nascondere al comando **ls** determinati files e cartelle
- nascondere processi a **ps** e fare in modo che non siano elencati in **/proc**
- nascondere connessioni di rete a **netstat**

Esempi di rootkit **LKM** per linux sono **ryal**, **heroin**, **afhrm**, **synapsis**, **adore**, **knark**, **itf**, **kis**, **adore-ng**. Riuscire a scovare tali rootkit diventa quindi piuttosto complesso rispetto ai classici che modificano i programmi di sistema.

Una volta avvenuta la compromissione potrebbe essere utile uno strumento quale **chkrootkit**, che consente di individuare l'avvenuta compromissione da parte dei principali rootkit conosciuti, compresi anche alcuni di tipo **LKM** quali **adore**, o **SuckIT**. Se pero' il rootkit installato dall'attacker non e' tra quelli riconosciuti il problema di accorgersi della sua presenza rimane irrisolto...

Una possibilita' potrebbe essere quella di eseguire un confronto tra gli indirizzi delle syscall del kernel con la mappa dei simboli e dei relativi indirizzi definiti nella **System.map** generata dopo la compilazione. Naturalmente tale file deve essere stato preventivamente copiato in un posto sicuro, prima prima della compromissione. Esistono alcuni software che potrebbero essere usati in tal senso, per rilevare modifiche al kernel quali **kern_check**, **checkIDT**, **Kstat** o **samhain**.

Alla luce dei grossi rischi a cui un kernel con il supporto per i moduli puo' esporre un server, mi sembra fondamentale disabilitare questa funzionalita', costruendosi un kernel completamente monolitico. In questo modo per un attacker non sara' possibile ricorrere all'utilizzo dei moduli allo scopo di alterare il nucleo del sistema.

Anche disabilitando il supporto ai moduli, esiste ancora una possibilita' che consente di installare rootkit che modifichino il kernel, con conseguenti problemi per identificarli, ovvero scrivere direttamente in memoria tramite i device **/dev/mem** e **/dev/kmem**. Tale sistema e' purtroppo sfruttabile anche in kernel non modulari.

kmem, e' un device che da accesso all'area di memoria occupata dal kernel in esecuzione. Scrivendo in quest'area e' possibile sovrascrivere il kernel a runtime, apportando quindi modifiche allo stesso. La difficolta' e' data dal trovare la corretta locazione entro cui scrivere.

Certamente subire questo tipo di modifica al kernel e' molto meno probabile, per il solo fatto che non tutti gli attacker possono essere sufficientemente abili da eseguire delle modifiche a runtime sul kernel. Esistono pero' anche rootkit che sfruttano questo sistema per installarsi sul sistema, quale ad esempio **SuckIT** (Fonte: phrack magazine n. 58: Linux on-the-fly kernel patching without LKM).

Una possibile soluzione a questo problema potrebbe essere una patch al kernel che ponga il device **kmem** in sola lettura. Alcuni programmi potrebbero non funzionare correttamente, ma e' un inconveniente accettabile se paragonato al rischio a cui ci esporrebbe in caso contrario. Per tale scopo e' stata utilizzata **grsecurity**, una patch, che tra le sue features contempla anche la possibilita' di porre tali devices in sola lettura.

Il kernel preparato, inoltre, non e' stato compilato direttamente su **paranoid**, ma su una macchina diversa, poiche' su quest'ultimo non erano presenti ne il compilatore **gcc**, ne le librerie dev richieste. La presenza di un compilatore su un server sarebbe secondo me un rischio da evitare, se possibile; da qui la scelta di compilare il kernel su un'altra macchina e copiarlo in seguito.

Entrando maggiormente nel dettaglio delle operazioni svolte, una volta scaricati i sorgenti da <http://www.kernel.org>, sono stati decompressi in **/usr/src** ed e' stato creato un link simbolico alla loro directory con il comando **ln -s /usr/src/linux /usr/src/linux-2.4.32**.

E' inoltre stata scaricata l'ultima versione di **grsecurity** dal sito <http://www.grsecurity.net>.

Decompresso il file in **/usr/src/linux**, da tale directory e' stata applicata la patch con il comando **patch -p1 < grsecurity-2.1.7-2.4.32-200511131857.patch**.

A questo punto, ho iniziato configurazione del kernel con il comando **make menuconfig**, scegliendo le singole opzioni da includere. Per motivi di spazio e di tempo non mi soffermo nel citarle tutte, ma solo quelle che ritengo piu' importanti:

- **Code maturity level options:** e' stata esclusa l'opzione **prompt for development and/or incomplete code/drivers**. Trattandosi di un server credo non sia il caso di includere funzionalita' ancora in fase di sviluppo, che potrebbero quindi essere causa di instabilita'.
- **Loadable module support:** si e' scelto di non includere nel kernel l'opzione **enable loadable module support**, in modo da impedire l'utilizzo ed il caricamento dei moduli, per i motivi precedentemente discussi. Disabilitando tale opzione, ora si potra' solo scegliere se includere **[y]** o meno **[n]** porzioni di codice all'interno del kernel. In caso contrario si sarebbe avuta anche la possibilita' di includerle come moduli **[m]**.
- **Processor type and features:** montando **paranoid** un pentium III e' stato scelto **PentiumIII/Celeron** come **processor family**, ed impostata ad off la voce **high memory support**, montando la macchina meno di 960 Mb di ram. E' stato disabilitato anche il **symmetric multi-processing support**, essendo la macchina monoprocesso.
- **General setup:** sono stato disabilitati l' **ACPI support** perche' ancora sperimentale secondo la documentazione fornita, **PCMCIA/CardBus support**, non avendo hardware di quel tipo, ed **ISA BUS support** perche' il server non monta alcuna schede di tipo ISA.
E' stato invece incluso **Support for hot-pluggable devices**, poiche' potrebbe servire nel caso si presentasse la necessita' di collegare dischi esterni tramite interfaccia usb.
- **Parallel port support:** e' stato disabilitato il supporto alla porta parallela, non avendo stampanti e non prevedendo l'utilizzo di altro hardware tipo lettori iomega zip che potrebbero essere connessi a tale porta.
- **Plug and play configuration:** e' stato incluso il supporto **Plug and Play configuration**, ed e' stato disabilitato **ISA Plug and Play Support**, non prevedendo l'utilizzo di schede che sfruttino il bus ISA.
- **Block devices::** e' stato incluso esclusivamente **Normal floppy disk support**, per consentire l'utilizzo del floppy disk.
- **Multi-device support:** utilizzando un solo hard disk, senza quindi sfruttare il supporto per il raid software, e' stata esclusa l'opzione **Multiple devices support (RAID and LVM)**
- **Networking options:** tra le varie opzioni incluse credo sia particolarmente importante ricordare **Network packet filtering**, che consente l'utilizzo di **netfilter** ed **iptables** per il filtraggio dei pacchetti, **TCP/IP networking** per la suite di protocolli **TCP/IP** e **TCP syncookie support**, per attivare la protezione verso attacchi di tipo Syn-flooding.
Tramite questa protezione lo stack **TCP/IP** usera' un challenge cryptographic protocol, conosciuto appunto come "**SYN cookies**", per consentire agli utenti legittimi di continuare a connettersi alla macchina anche se essa e' sottoposta ad un tentativo di attacco DoS, il tutto in maniera trasparente per loro ed il rispettivo software **TCP/IP** (fonte: <http://cr.yp.to/syncookies.html>).
Si ricorda inoltre, che tale protezione non viene attivata per default, quindi l'operazione deve essere effettuata una volta avviata la macchina intervenendo sull'opportuno file in **/proc** con il comando: **echo 1 > /proc/sys/net/ipv4/tcp_syncookies**.
Non vengono trattate le singole opzioni incluse nella sottosezione **IP:Netfilter configuration**, poiche' l'argomento della relazione e' la sicurezza locale dal sistema, e non i dettagli della configurazione di un firewall basato su **netfilter**.
- **Telephony Support:** non e' stata inclusa l'opzione **Linux Telephony Support**, non avendo hardware che lo

richieda, quali schede che consentono l'uso di un telefono per applicazioni **VOIP**.

- **ATA/IDE/MFM/RLL support:** e' stata inclusa l'opzione **ATA/IDE/MFM/RLL support**, non avendo periferiche e controller di tipo **SCSI**. In caso si utilizzassero invece esclusivamente dischi, lettori, controller ecc. di tale tipo, sarebbe stato possibile disabilitare questa opzione. Nella sottosezione **IDE, ATA and ATAPI Block devices** ho invece incluso esclusivamente le voci relative al mio hardware, e disabilitato la voce **use multimode by default**.
- **SCSI support:** ho incluso l'opzione **SCSI support**, pur non avendo hardware di questo tipo sul server, poiche' necessaria per il supporto ai dischi USB esterni.
- **I2O device support:** esclusa l'opzione **I2O device support** non avendo schede adapter di questo tipo
- **Amateur radio support:** e' stata esclusa l'opzione **Amateur radio support**, non avendo la necessita' di connettere la macchina ad apparecchiature radioamatoriali.
- **IrDA support:** e' stato escluso **IrDA subsystem support**, non avendo periferiche wireless ed infrarossi.
- **ISDN subsystem:** e' stato escluso **ISDN support**, non utilizzando schede e linee ISDN.
- **Multimedia device:** esclusa l'opzione **Video for linux**, non avendo periferiche per la cattura audio/video, o schede per radio FM.
- **Sound:** e' stata esclusa l'opzione **Sound card support**. Sebbene sulla scheda madre fosse presente una scheda audio integrata, non ho ritenuto potesse essere utile per un sistema server, e quindi ho disattivato direttamente il supporto nel kernel.
- **Bluetooth support:** e' stata esclusa l'opzione **Bluetooth subsystem support**, non utilizzando simili periferiche .
- **Network device support:** ovviamente e' stata inclusa la voce **network device support**, poiche' devono essere possibili connessioni tra **paranoid** ed altre macchine. E' stata inoltre abilitata anche la voce **Universal TUN/TAP device driver support**, poiche' necessaria per la realizzazione della VPN utilizzata nel trasferimento dei log tra **paranoid** e **stargazer**.
Nella sottosezione **ethernet (10 or 100 Mbit)** e' stata attivata l'opzione omonima, che consente il supporto alle schede 10/100 . Sempre in questa sezione, sono stati inclusi **EISA, VLB, PCI and on board controllers** e **Realtek RTL-8139 Fast Ethernet Adapter Support**, poiche' relativi alla scheda di rete con cui il pc e' equipaggiato.
E' stato inoltre tolto dal kernel il supporto al Point to Point protocol, escludendo l'opzione **PPP support**, poiche' il server si trova dietro ad un router che si occupa di gestire la connessione, e quindi non si prevede la necessita' di creare una connessione punto punto, come ad esempio avrebbe potuto essere necessario utilizzando un modem.
- **Character devices:** sono stati incluse le voci **virtual terminal, support for console on virtual terminal** ed **UNIX pty 98 support**, per potere utilizzare un emulatore di terminale sulle diverse tty, e quindi potere operare sul sistema.
Nella sottosezione **mice** sono state incluse invece le voci **Mouse support** e **PS/2 mouse support**.
E' inoltre stata esclusa dal kernel l'opzione **/dev/agpgart (AGP support)**, perche' sebbene il sistema monti una

scheda grafica agp, non e' previsto l'utilizzo e neppure l'installazione del server X, gestendo interamente il sistema da console. Allo stesso modo e' stato disabilitato il **Direct Rendering Manager (Xfree86 DRI support)**.

- **File systems:** e' stato incluso il supporto delle quote disco (**Quota support** e **VFS v0 quota format support**). E' inoltre stato incluso il supporto ai principali tipi di filesystem: **ext2**, **ext3**, **ReiserFs**, **Xfs**, **Jfs**, **DOS FAT**, **VFAT**, **ISO9660**, **NTFS** (sola lettura), **UDF** ed **UFS**. Il requisito minimo sarebbe stato l'inclusione del solo **ext3**, **ISO9660** ed **UDF**, per accedere alle partizioni del disco e leggere cd e dvd . Ho preferito pero' includere anche il supporto agli altri filesystem nel caso dovesse presentarsi la necessita' di montare dischi con filesystem differenti, sebbene non strettamente indispensabile.

Nella sottosezione **network filesystems** sono state abilitate le voci relative al supporto per **NFS (NFS file system support** e **Provide NFSv3 client support**) e Samba (**SMB filesystem support**). Come si puo' notare e' stato abilitato il solo supporto client, nel caso si presenti la necessita' di connettersi a condivisioni NFS o samba. Presumendo che l'host non debba mai svolgere la funzione di server NFS e' invece stata esclusa l'opzione **NFS server support**.

Nella sottosezione **Partition types**, sono state incluse le opzioni **Advanced partition selection**, e **PC BIOS (MSDOS partition table) support**. Tale scelta e' stata necessaria perche' il disco fisso e' stato partizionato su un sistema della famiglia i386. In caso contrario, il kernel ottenuto non sarebbe riuscito ad avviare correttamente il sistema, riportando un Kernel Panic relativo al VFS.

- **Console drivers:** sono state incluse le opzioni **VGA text console**, per potere utilizzare la console testuale su un sistema che supporti lo standard VGA, e **Video mode selection support**, per potere specificare tra i parametri di boot il numero di linee e la dimensione del testo in console.
- **USB support:** sono stati incluse le opzioni **Support for USB**, ed **UHCI alternate driver support**, avendo un controller **USB** di tipo **UHCI**. Inoltre, essendo interessato alla possibilita' di usare dischi esterni e pen-drive e' stato inclusa anche **USB Mass storage support**.
- **Cryptographic options:** sono state incluse **Cryptographic API** e le voci relative agli algoritmi di message digest **SHA**, **MD5** e gli algoritmi di cifratura **AES**, **Blowfish**, e **Twofish**.

La prima parte della configurazione del kernel puo' essere considerata completata, ma manca ancora la voce relativa alla patch **grsecurity**. Prima di entrare nel dettaglio delle opzioni incluse, e' pero' necessario fare alcune considerazioni relative al suo utilizzo e soprattutto ai motivi per cui vi si faccia ricorso.

Infatti, nonostante tutte le considerazioni fatte e le misure prese per aumentare la sicurezza locale del sistema, esso resta ancora esposto a pericoli estremamente gravi, legati agli errori di programmazione.

Sfruttando ad esempio dei buffer overflow un attacker potrebbe cercare di sovrascrivere il **return address** all'interno dello stack, facendo in modo che vada a puntare ad un'altra area di memoria, facendogli eseguire dello specifico codice, come ad esempio l'apertura di una shell. Così facendo l'esecuzione proseguirebbe normalmente dal nuovo indirizzo indicato nel **return address**, aprendo quindi una shell con i privilegi dell'utente che aveva lanciato il programma.

Si possono immaginare gli effetti disastrosi di un buffer overflow sfruttato correttamente su un programma che gira con i privilegi dell'utente **root**.

Molti di questi errori vengono rapidamente corretti dai diversi team di sviluppo dopo le prime segnalazioni e proof of concept sui principali siti dedicati alla sicurezza. Si presume pero' che esistano anche diversi zero-days basati su questa classe di vulnerabilita', e quindi ancora piu' subdoli, poiche' essendo sconosciuti si ignora di essere vulnerabili.

Una possibile soluzione a questo tipo di vulnerabilit  e' la cosiddetta **Address Space Layout Randomization (LASR)**. Poich  attacchi di questo tipo richiedono considerazioni relative agli indirizzi di memoria dell'applicazione da attaccare, l'idea sarebbe quella di introdurre degli elementi di casualit , facendo in modo che lo spazio di indirizzamento cambi in modo imprevedibile ogni volta che un processo viene lanciato.

In questo modo un attacker deve anche indovinare dove si trovi il **return address**, riducendo l'attacco ad un brute-force. Se la disposizione degli spazi di memoria fosse quindi sufficientemente "casuale" diventerebbe impossibile per l'attacker portare a termine attacchi di questo tipo.

Anche con questa soluzione, un'applicazione mal programmata potrebbe andare incontro ad un Segmentation Fault, ma quantomeno non verrebbe data all'attacker la possibilit  di eseguire codice arbitrario.

Esiste appunto un patch per il kernel Linux, chiamata **PaX**, che implementa questo tipo di protezione. Essa puo' essere usata separatamente, oppure la si puo' trovare inclusa in un'altra patch di maggiori dimensioni: **grsecurity**. Ho optato per l'utilizzo di **grsecurity**, perch  consente anche la possibilit  di porre i device **/dev/mem** e **/dev/kmem** in sola lettura, arginando quindi anche rootkit tipo **SuckIT**.

Le varie opzioni relative a questi aspetti della configurazione si trovano nel menu' **grsecurity** della configurazione del kernel:

- **Grsecurity**: e' stata attivata tale opzione, poich  e' requisito essenziale per scegliere le voci sottostanti nella sottosezione omonima.
- **Security Level**: e' possibile scegliere tra diversi livelli di protezione predefiniti. Ho optato per **customized** in modo da avere scelta libera nelle soluzioni da adottare.
- **Pax Control**: e' stata inclusa l'opzione **Use elf program header marking**, per consentire successivamente la possibilit  di modificare il comportamento di **PaX** per specifici eseguibili tramite l'utility **paxctl**. E' inoltre stata inclusa l'opzione **Use legacy elf header marking**, per consentire la protezione anche delle applicazioni non marcate con uno specifico elf program header per **PaX**. Ho infatti avuto modo di appurare tramite il comando **paxtest** come il sistema risultasse ancora vulnerabile utilizzando un kernel privo di quest'ultima opzione.
- **Address space protection** : sono state incluse le voci **Enforce Non-executable pages**, **Paging-based non executable pages** e **Segmentation based non-executable pages**, per impedire tentativi di code injection da parte di eventuali attacker, ovvero introduzione di codice arbitrario nell'area di memoria di un processo.
Sfruttando alcuni errori di programmazione e' possibile introdurre codice arbitrario in alcuni punti della memoria del programma oggetto dell'attacco, tipicamente nello stack o nell'heap, ed eseguirlo. Ovviamente se tale programma stava girando con privilegi differenti, l'attacker guadagner  i privilegi dell'utente che aveva eseguito tale programma.
Poich  utilizzando il kernel Linux standard, una pagina di memoria   leggibile, come lo stack o lo heap ad esempio, e' anche eseguibile, e' chiara la necessit  di proteggersi. A tale scopo sono state incluse le voci citate relativamente alla **Address space protection** :

- **Enforce Non-executable pages**: consente di attivare ulteriori opzioni per prevenire l'iniezione e l'esecuzione di codice arbitrario in un programma. Attivando tale meccanismo pero', alcuni programmi che fanno un uso particolarmente "artistico" della funzione **malloc()** non saranno in grado di funzionare, come ad esempio il server Xfree86, java runtime ecc.. Nel mio caso non essendo presente nessuno di tali software il problema non si e' presentato, ma in caso di necessit  sarebbe possibile modificare i flag di **PaX** per gli eseguibili dove sarebbe necessario, tramite il comando **paxctl**.
- **Paging-based non executable pages** e **Segmentation based non-executable pages** sono basate su due features di paginazione e segmentazione della CPU. Secondo la documentazione, con cpu della famiglia

i386 tali opzioni potrebbero avere un impatto negativo sulle performance, a differenza che con processori parisc, sparc, ed alpha. Io ho scelto di attivare entrambe le opzioni, pero' in un ambiente di produzione la scelta dovrebbe prevedere anche un adeguato periodo di test, per vedere come ne risentirebbe il comportamento delle applicazioni che dovrebbero girare sul server.

- **Restrict mprotect()** . Abilitando tale opzione si impedisce ai programmi di: cambiare lo status delle pagine di memoria da non eseguibili ad eseguibili, rendere scrivibili pagine eseguibili, ma per le quali e' concesso solo l'accesso in lettura, e creare pagine eseguibili dall'anonymous memory (stack, heap ecc.).

Una seconda serie di voci estremamente impotanti in questa sottosezione e' quella relativa alla randomizzazione dello spazio di indirizzamento:

- **Address Space Layout Randomization**: attivando questa voce viene consentito di selezionare delle ulteriori opzioni relative all'argomento.
- **Randomize kernel stack base** consente di introdurre un elemento di casualita' nel kernel stack pointer dei processi. Linux assegna ad ogni task due pagine di kernel stack, che viene usato ogni volta che il processo "entra" in kernel-space ad esempio eseguendo una system call, generando interrupt, eccezioni ecc. Eseguita l'azione il processo ritorna in user-space. Tale soluzione, cambia la randomizzazione ad ogni system call.
- **Randomize user stack** invece introduce casualita' negli indirizzi dello stack userspace del processo. Ogni task ha uno stack userspace, creato durante **execve()** e copiato sui processi figli durante le **fork()**. Attraverso questo stack il kernel passa gli argomenti e variabili d'ambiente ai task.
Normalmente, esso viene creato in due passaggi dal kernel, al termine dello spazio di indirizzamento userland, in modo che possa crescere verso il basso in caso di necessita'. Nel primo, il kernel alloca e riempie le pagine per lo stack, mentre nel secondo le mappa nello spazio di indirizzamento del processo.
Il primo intervento viene fatto nella funzione **do_execve()** (definita in **fs/exec.c**), nella randomizzazione di alcuni bit in uno stack pointer temporaneo utilizzato dal kernel per tenere traccia di quanto copiato nelle pagine. La seconda modifica e' invece nella funzione **setup_arg_pages()**, che si occupa di mappare la pagine di stack create nello spazio di indirizzamento del processo.
- **Randomize mmap() base** introduce casualita' nelle regioni di memoria gestite tramite la funzione **do_mmap()**. In questo modo le librerie caricate dinamicamente si troveranno in indirizzi di memoria casuali, rendendo piu' difficile per un attacker tentare di eseguire codice da queste librerie all'interno di un exploit.

Sempre in questa sezione, sono infine state incluse le voci **Deny writing to /dev/kmem, /dev/mem and /dev/port** per i motivi precedentemente discussi e **Remove Addresses from /proc/pid/[maps | stat]**, per non visualizzare informazioni sugli indirizzi di memoria del processo nei files **/proc/<pid>/maps** e **/proc/<pid>/stat**.

- **Filesystem protections**: in questa sottosezione sono state incluse le seguenti opzioni
 - **Proc restrictions**: vengono modificati i permessi di default all'interno del proc- filesystem.
 - **Restrict to user only**: in questo modo un utente potra' vedere solo i propri processi all'interno di **/proc**. Inoltre gli verra' impedito di visualizzare statistiche ed informazioni relative alla rete, i simboli del kernel, e informazioni sui moduli.
 - **Additional restrictions**: vengono poste delle restrizioni addizionali agli utenti relativamente al proc-filesystem, in modo da impedire la visualizzazione di informazioni relative ai vari device ed al file **slabinfo**, che potrebbero essere utili per eventuali tentativi di compromissione. fornendo una fotografia

dello stato d'uso della memoria.

- **Linking restrictions:** impedisce agli utenti di seguire link simbolici di proprietà di altri utenti, e posti all'interno di directory word-writable, a meno che il proprietario del link non lo sia anche per la directory. Viene inoltre negata la possibilità di creare hard link a file di cui l'utente non sia proprietario.
- **FIFO restrictions:** impedisce agli utenti di scrivere su file speciali FIFO all'interno di directory word-writable (ad esempio **/tmp**), a meno che il proprietario del file non sia lo stesso della directory entro cui è esso contenuto.
- **Kernel auditing:** in questa sottosezione sono state incluse le seguenti opzioni
 - **Resource logging:** consente di registrare nei log tutti i tentativi di superare eventuali limiti imposti per l'utilizzo delle risorse, indicando nome della risorsa, quantità richiesta e attuale limite.
 - **(Un)Mount logging:** consente di loggare tutti i mount ed umount effettuati
 - **Signal logging:** consente di loggare l'invio di segnali particolarmente importanti, tipo **SIGSEGV**. Nell'help e nella documentazione disponibile non è però menzionato quali siano gli altri segnali loggati.
 - **Fork failure logging:** consente di loggare i tentativi di **fork()** falliti. Una simile soluzione potrebbe essere utile per riportare i tentativi di forkbomb da parte di qualche utente.
 - **Time change logging:** registra le modifiche all'orologio di sistema. Un eventuale attacker potrebbe per esempio cercare di modificare data e ora, per consentire l'accesso ad un determinato servizio, eludendo eventuali limitazioni predisposte dall'amministratore di sistema con il modulo **pam_time**.
- **Executable protections:** in questa sottosezione tra le voci incluse le più importanti sono state:
 - **Enforce RLIMIT_NPROCS on exec:** normalmente il limite sul numero di processi per un utente viene controllato solo durante le chiamate alla funzione **fork()**. Attivando questa opzione il controllo avviene anche per **execve()**.
 - **Dmesg restriction:** includendo questa opzione si impedisce agli utenti di visualizzare i gli ultimi 4Kb. di messaggi nel log buffer del kernel, tramite il comando **dmesg**.
 - **Randomized PIDs:** consente di generare i PID dei processi in maniera pseudocasuale. Abbinando questa opzione alle limitazioni sul filesystem proc, sarà estremamente difficile per un attacker riuscire ad indovinare i PID dei demoni e dei processi in generale. Poiché i PID vengono talvolta utilizzati come porzioni dei nomi di alcuni files temporanei, generandoli casualmente, sarà più difficile anche prevedere i nomi di tali files.
- **Network protections:** è stata attivata l'opzione **Randomize TCP source ports**, per fare in modo che vengano generati casualmente i numeri di porta sorgente nei pacchetti TCP, anziché utilizzando un semplice algoritmo che li incrementi ad ogni nuova connessione.

Completata la fase di configurazione del kernel, prima di iniziare la compilazione sono state create le strutture delle dipendenze per il compilatore e rimossi alcuni file temporanei e di backup con i comandi **make dep** e successivamente **make clean**.

La compilazione vera e propria e la generazione dell'immagine del kernel è invece stata eseguita tramite il comando **make bzImage**.

Terminata la fase di compilazione sono stati copiati nella directory **/boot** di **paranoid** i files **/usr/src/linux/System.map** e **/usr/src/linux/arch/i386/boot/bzImage**, rinominandoli in **System.map-2.4.32-GSEC** e **vmlinux-2.4.32.GSEC**.

Si è quindi modificato il file **/etc/lilo.conf** in modo da aggiungere una nuova voce al menu, ed impostare

l'immagine del nuovo kernel come default:

```
default=linux_hardened
#Linux + Grsecurity
image=/boot/vmlinuz-2.4.32-GSEC
label=linux_hardened
password=P5ych0t1Cwaltz
restricted
read-only
append="rootflags=data=journal"
```

Per sicurezza si e' scelto di mantenere anche la vecchia immagine, in modo da avere la possibilita' di avviare il sistema anche in caso di kernel panic oppure altri problemi con la nuova versione. Si sono quindi scritte le nuove modifiche nel master boot record tramite il comando **lilo** ed e' stato riavviato il sistema, selezionando al boot la nuova label **linux_hardened**.

Fortunatamente la procedura e' stata portata a termine correttamente, senza alcun kernel panic, e si e' potuto verificare che il sistema stesse effettivamente girando con il nuovo kernel, digitando il comando **uname -r**. Correttamente, l'output di tale comando e' stata la stringa **2.4.32-grsec**, poiche' nel Makefile del kernel era stata appunto specificata la stringa -grsec per l'opzione **EXTRAVERSION**.

Per testare i meccanismi di protezione della memoria si e' invece utilizzata un'utility apposita, **paxtest**, installata con il comando **apt-get install paxtest**. Con un kernel tradizionale il risultato dei vari test effettuato dal programma sarebbe stato **Vulnerable**, senza riportare inoltre alcuna randomizzazione dello stack.

Con il kernel preparato, invece, il risultato del comando **paxtest blackhat** e' stato il seguente:

```
PaXtest - Copyright(c) 2003,2004 by Peter Busser <peter@adamantix.org>
Released under the GNU Public Licence version 2 or later
```

```
Mode: blackhat
```

```
Linux paranoid 2.4.32-grsec #2 Sat Jan 7 16:25:52 CET 2006 i686 GNU/Linux
```

```
Executable anonymous mapping      : Killed
Executable bss                    : Killed
Executable data                   : Killed
Executable heap                   : Killed
Executable stack                  : Killed
Executable anonymous mapping (mprotect) : Killed
Executable bss (mprotect)         : Killed
Executable data (mprotect)        : Killed
Executable heap (mprotect)        : Killed
Executable shared library bss (mprotect) : Killed
Executable shared library data (mprotect): Killed
Executable stack (mprotect)       : Killed
Anonymous mapping randomisation test : 16 bits (guessed)
Heap randomisation test (ET_EXEC)  : 13 bits (guessed)
Heap randomisation test (ET_DYN)   : 25 bits (guessed)
Main executable randomisation (ET_EXEC) : No randomisation
```

Main executable randomisation (ET_DYN)	: 17 bits (guessed)
Shared library randomisation test	: 16 bits (guessed)
Stack randomisation test (SEGMEEXEC)	: 23 bits (guessed)
Stack randomisation test (PAGEEXEC)	: 23 bits (guessed)
Return to function (strcpy)	: Vulnerable
Return to function (strcpy, RANDEEXEC)	: Vulnerable
Return to function (memcpy)	: Vulnerable
Return to function (memcpy, RANDEEXEC)	: Vulnerable
Executable shared library bss	: Killed
Executable shared library data	: Killed
Writable text segments	: Killed

I test della serie **Executable**, cercano di inserire istruzioni nella **DATA** region del processo ed eseguirle, mentre **Writable text segments** tenta di sovrascrivere della memoria marcata come eseguibile. Il risultato **killed** sta ad indicare che il kernel ha bloccato i tentativi di sfruttare in maniera la memoria.

Quattro test risultato vulnerabili, ovvero i **Return to function**. Secondo quanto letto nella mailing list del progetto **grsecurity**, tali risultati sono da ritenersi normali, perche' legati al compilatore, e risolvibili ricorrendo ad estensioni dei compilatori GNU quale **SSP** (Stack-Smashing protector) o **StackGuard**.

StackGuard e' un estensione commerciale, e l'idea su cui e' basato e' l'introduzione di una "canary word" prima del **return address** nello stack. In questo modo, quando una funzione termina, si controlla la validita' della "canary word", scoprendo o meno se il buffer e' stato sovrascritto. In caso di sovrascrittura il programma viene semplicemente terminato.

SSP invece mette il buffer dopo il **return address** e lo **stack frame pointer**, in modo da evitare la possibilita' di sovrascriverli e quindi la loro corruzione (**fonte** : <http://www.research.ibm.com/trl/projects/security/ssp/>).

Questo genere di protezioni, pero', vengono "inserite" in un'applicazione direttamente durante la compilazione, quindi per proteggere i diversi eseguibili, essi andrebbero tutti ricompilati con una di questa estensione del **gcc**, una soluzione forse un po' drastica in molti casi...

Bibliografia

- Amir Malik : Filesystem security
- [Http://www.campin.net](http://www.campin.net) : Syslog-ng FAQ
- [Http://www.courtesan.com](http://www.courtesan.com) : Sudoers manual
- [Http://cr.yp.to/syncookies.html](http://cr.yp.to/syncookies.html) : Syn cookies
- Daniel Robbins : surprises in ext3 (<http://www-128.ibm.com/developerworks/linux/library/l-fs8.html#4>)
- [Http://www.debianizzati.org](http://www.debianizzati.org) : guida a OpenVpn
- [Http://docs.hp.com](http://docs.hp.com) : Disk array che usano le strategie di protezione dei dati RAID
- Francesco Di Rienzo, Gianluca Esposito, Feliciano Nigro : Syslog-ng, manuale d'uso
- [Http://www.gentoo.it](http://www.gentoo.it) : guida di SUDO(ers)
- Gentoo Linux wiki: Security – Limit users processes
- Gentoo security handbook : mounting partitions
- Gentoo security handbook : file permissions
- Gentoo security handbook : Logging
- Gentoo security handbook : PAM
- Gentoo security handbook : user/groups limitations
- Grsecurity quickstart guide
- [Http://www.grsecurity.net/wiki/index.php](http://www.grsecurity.net/wiki/index.php) : grsecurity wiki
- [Http://it.wikipedia.org](http://it.wikipedia.org) : Redundant Array of Independent Disks
- Kurt Seifried : limiting and monitoring users
- Kurt Seifried : Linux installation, Filesystem layout and structuring
- Kurt Seifried : physical and console security
- [Http://la-samhna.de](http://la-samhna.de) : Linux kernel rootkits
- Linas Vepstas : Software-RAID HOWTO
- [Http://www.linsec.ca](http://www.linsec.ca) : AIDE, Advanced Intrusion Detection Environment
- [Http://www.linsec.ca](http://www.linsec.ca) : Using Sudo to limit access
- [Http://linux.codeso.be](http://linux.codeso.be) : syslog-ng examples
- Linux&C. n. 34, Luigi Genoni : Filesystem journaled, robustezza e prestazioni
- Linux&C. n. 36, Claudio Panichi : Hardening Linux, come blindare il vostro server
- Linux&C. n. 37, Claudio Panichi : rendere sicuro il filesystem
- Linux&C. n.38, Claudio Panichi : Loadable Kernel Modules, ne abbiamo veramente bisogno?
- Linux&C. n.39, Claudio Panichi : Integrita' del sistema
- Linux&C. n.40, Claudio Panichi : Pluggable Authentication Modules
- Linux&C. n. 43, Claudio Panichi : Restrizione dei privilegi locali
- Linux&C. n. 44, Claudio Panichi : proteggersi dagli errori di programmazione
- Linux&C. n. 49, Claudio Panichi : avviare i servizi con inetd e rinetd
- Linux Administrator's Security Guide : Filesystem security
- Linux Security HOWTO : Sicurezza fisica
- Michael D. Bauer, Building secure servers with Linux : System log management and monitoring
- [Http://www.openskills.info](http://www.openskills.info) : Hardening con grsecurity e kernel 2.4.29
- [Http://www.openskills.info](http://www.openskills.info) : il backup
- [Http://www.openskills.info](http://www.openskills.info) : Intrusion Detection Systems (IDS) su Linux
- [Http://www.openskills.info](http://www.openskills.info) : linux e la sicurezza fisica
- [Http://www.openskills.info](http://www.openskills.info) : Troubleshooting di programmi e permessi sui file
- Paolo Pavan : Backup sotto Linux
- [Http://pax.grsecurity.net/docs/index.html](http://pax.grsecurity.net/docs/index.html) : documentazione PaX

- Phrack magazine n. 58, sd@sf.cz, devik@cdi.cz : Linux on-the-fly kernel patching without LKM
- Phrack magazine n. 59, kadamyse@altern.org : handling the interrupt descriptor table
- Pluto journal n. 29 , Eugenia Franzoni : I backup
- Pluto journal n. 35, Germano Rizzo : Il pinguino legge il giornale
- Pluto journal n. 45, Vincenzo Giacchina : OpenVPN, VPN in pochi passi
- Red Hat Linux 7.3 Official Red Hat Linux Reference Guide : Panoramica sull'FHS
- [Http://www.research.ibm.com/trl/projects/security/ssp](http://www.research.ibm.com/trl/projects/security/ssp) : GCC extension for protecting applications from stack-smashing attacks
- [Http://rfc.sourceforge.net](http://rfc.sourceforge.net) : Remote Filesystem Checker
- RFC 3164 - The BSD Syslog Protocol
- Securing and Optimizing Linux, RedHat Edition -A Hands on Guide : Automating backups with tar
- SecurityFocus-Linux mailing list : SUID program removal
- [Http://sicurezza.html.it](http://sicurezza.html.it) : Logcheck: scoprire le intrusioni attraverso i file di log
- Werner Puschitz : Securing Linux production system
- Yvette Agostini, Valerio Verde : File di log, importanza e analisi
- [Http://www.ziobudda.net](http://www.ziobudda.net) : Sicurezza di Linux, consigli pratici