ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Bari

INFN/TC-04/08
14 Aprile 2004

REALIZZAZIONE DI UN WEB SERVER SICURO CON USER MODE LINUX

Domenico Diacono, Giacinto Donvito

INFN-Sezione di Bari, Via Orabona 4, I-70126 Bari, Italy

Abstract

La realizzazione di un Web server è sempre più spesso necessaria nella attività di ricerca per permettere la condivisione rapida di informazioni. La diffusione capillare dell'accesso alla rete e il conseguente aumento dei casi di propagazione di virus e worm che sfruttano vulnerabilità nei server impone una forte attenzione alla sicurezza.

In questo studio si propone l'uso di una distribuzione GNU/Linux poco diffusa, Gentoo-Linux, all'interno della quale mediante la modifica del kernel User Mode Linux attivare un server virtuale completamente separato dalla macchina ospite.

In questo modo una eventuale compromissione attraverso i servizi Web della macchina virtuale avrà scarse probabilità di raggiungere la macchina principale.

Published by ***SIS–Pubblicazioni***
Laboratori Nazionali di Frascati

SOMMARIO

1 INTRODUZIONE

La realizzazione di un Web server è sempre più spesso necessaria nella attività di ricerca per permettere la rapida condivisione di informazioni. La diffusione capillare dell'accesso alla rete e il conseguente aumento dei casi di propagazione di virus e worm che sfruttano vulnerabilità nei server impone però una forte attenzione alla sicurezza.

In questo studio si propone l'uso di una distribuzione GNU/Linux attualmente poco diffusa, *Gentoo Linux,* all'interno della quale mediante la modifica del kernel *User Mode Linux* attivare un server virtuale completamente separato dalla macchina ospite. Questa precauzione consente l'isolamento della parte web-server dal sistema operativo. La procedura illustrata ha carattere generale, ed è applicabile ad una qualsiasi distribuzione.

Nel seguito ai comandi da digitare direttamente nella shell GNU/Linux verrà premesso il simbolo **#** se si tratta di comandi di root, **$** se si tratta di comandi da eseguire come utente non privilegiato; il server reale sul quale si è installato il web server verrà detto *host* o macchina ospite, il server virtuale verrà chiamato *uml* o macchina UML.

1.1 Le regole fondamentali della sicurezza

Innanzitutto è necessario essere consapevoli che la sicurezza di una macchina non è un traguardo statico che può essere raggiunto una volta per tutte, ma piuttosto si tratta di uno stato dinamico nel quale la macchina viene continuamente modificata in risposta a nuovi pericoli e nuove informazioni, derivanti ad esempio dall'analisi dei tentativi di intrusione.

In appendice A viene riportata una analisi a grandi linee delle regole alla base di ogni decisione riguardante la sicurezza[1].

Nella costruzione di un Web Server, a parte le considerazioni sulle misure di sicurezza generali che scaturiscono dalla applicazione delle ultime regole, si può osservare in particolare che:

1. Non devono essere attive applicazioni non connesse strettamente con l'attività di web server, ed anzi si deve puntare ad avere il minor numero possibile di pacchetti software.
2. Se sono gestite delle pagine personali gli utenti devono avere la possibilità di accedere esclusivamente alle loro directory web personali.
3. L'accesso con login interattivo deve essere concesso solo se strettamente necessario.

Nel caso in esame non è possibile negare l'accesso interattivo, e si deve offrire un sistema nel quale l'utente ha una autonomia difficilmente conciliabile con l'ideale isolamento della macchina.

1.2 Sicurezza di base in sistemi multiutente

La appendice B riporta una lista, dedicata in particolare ad un web server, di precauzioni basilari che eliminano o perlomeno riducono la possibilità che tool automatici molto semplici

e diffusi possano compromettere il sistema, anche con l'aiuto inconsapevole di un utente. La lista è certamente non esaustiva, ma copre molti aspetti spesso trascurati.

Le precauzioni indicate naturalmente non esauriscono le necessità di sicurezza di un web server, tenuto conto anche del fatto che in un ambiente di ricerca molte di queste non possono essere adottate. Bisogna quindi considerare almeno l'adozione di:

- Un sistema di analisi automatica dei log. L'analisi manuale dei file può essere faticosa, tanto da risultare non praticabile giornalmente. In questo compito possono essere di aiuto dei programmi che analizzano i log di sistema, evidenziando errori e avvisi ed inviandoli immediatamente via mail o sms all'amministratore. Un esempio di software di tale genere è Logcheck. [2]
- Un sistema di rivelazione delle intrusioni, che permetta di sapere se la macchina ha subito manomissioni nelle sue componenti essenziali. Un esempio di software di tale genere è Tripwire. [3]
- Un firewall che chiuda l'accesso a tutte le porte TCP ed UDP non utilizzate dal sistema. Il kernel linux incorpora il firewall iptables [4].

L'applicazione di tutte le precauzioni descritte, sebbene possa ridurre di molto la possibilità di una intrusione non autorizzata, certamente non risolve il problema delle vulnerabilità degli applicativi. Una versione del web server Apache afflitta da un problema di sicurezza rischia cioè di compromettere seriamente la sicurezza dell'intero sistema.

Una prima risposta a queste esigenze può venire dall'inclusione del web server in quella che si chiama una "chroot jail" (cella chroot). La funzione *chroot()* è una chiamata di sistema che spesso viene utilizzata per fornire un grado di sicurezza aggiuntivo quando si eseguono programmi non fidati. Il kernel che supporta chroot() mantiene nota della directory di root che ogni processo ha sul sistema. Generalmente si tratta di "/", ma la chiamata chroot() può cambiarla: il processo assumerà quindi che la directory root sia quella data come argomento alla funzione chroot(). Ad esempio, dopo aver eseguito il codice seguente:

```
chdir("/foo/bar");
chroot("/foo/bar");
```

il processo vedrà la directory /foo/bar come la sua directory radice. La chiamata chroot() permette quindi di eseguire un server con un filesystem di root diverso da quello del sistema originario. In questo modo un utente è impossibilitato ad arrecare danni al sistema principale, e si possono limitare i danni derivanti dalla intrusione nel server.

Ci sono però una serie di problemi: innanzitutto l'utente root dell'ambiente chroot coincide con l'utente root del sistema ospite. Quindi se per un problema del server un attaccante riesce ad ottenere nell'ambiente chroot i privilegi di root può abbastanza facilmente scappare dalla cella e raggiungere il sistema ospite. Inoltre un utente root all'interno di chroot ha pieno controllo sulla attività di rete del sistema, quindi può mettere un suo server in ascolto o usare la macchina per un attacco DOS.

La manutenzione di un web server di ricerca all'interno di una cella chroot promette di essere molto complicata: ogni volta che deve essere aggiunta una nuova funzionalità è necessario individuare tutti i file coinvolti e copiarli all'interno della cella. La cosa può rivelarsi meno banale di quanto sembri dato che anche una semplice chiamata ad *ls* coinvolge quattro librerie, come si può verificare con:

```
$ldd /usr/bin/ls
```

Copiare tutto il sistema o gran parte di esso nella chroot jail del resto è improponibile: quante più risorse sono disponibili, tanto più facile sarà "scappare" dalla cella, a causa delle vulnerabilità del software. Un processo all'interno di chroot può ancora accedere a risorse al di fuori dell'ambiente che non siano file, ad esempio descrittori di area di memoria condivisa o socket, e quindi una cella non può proteggere dalla cooperazione di due processi, uno contenuto in essa e uno al di fuori di essa. Sono quindi necessari altri sistemi di controllo sul traffico dei socket.

Inoltre se è vero che per aumentare la sicurezza è necessario includere all'interno della prigione il minimo indispensabile, è anche vero che l'utente del web server prima o poi vorrà avere il controllo totale dei suoi file sul server, cosa che necessita di una shell, primo strumento necessario ad un attaccante per lasciare la cella. La sicurezza quindi può rivelarsi impraticabile nel caso in esame, e più in generale in ogni caso in cui la cella debba essere accessibile direttamente da utenti.

In questa ricerca del sistema più adatto ad isolare l'attività del server dal sistema operativo ospite ci si è imbattuti in User Mode Linux.

1.3 User Mode Linux

User-Mode Linux [5] fornisce una macchina Linux virtuale completa che gira all'interno della macchina fisica principale. Gli utenti di UML sono completamente separati dalla quelli della macchina *host*, e i processi all'interno del kernel UML non hanno accesso alla macchina fisica e al mondo esterno a meno che non glielo si garantisca esplicitamente. Normalmente i programmi in kernel space comunicano direttamente con l'hardware, e forniscono ai programmi in user space la possibilità di usare le risorse della macchina. Nel caso di UML il kernel viene eseguito in user space, e quindi l'accesso all'hardware non viene dato direttamente, ma attraverso il kernel ospite, così come accade per ogni altro programma. I programmi poi possono girare in user space all'interno della macchina UML come in una macchina qualsiasi.

Dato che manca l'accesso all'hardware UML non usa direttamente la memoria fisica o lo spazio disco: questo ad esempio può essere contenuto all'interno di uno o più file. E' possibile in altre parole assegnare alla macchina UML esclusivamente l'accesso hardware del quale ha realmente bisogno, limitando così la possibilità che la macchina virtuale danneggi la

macchina fisica o il suo software.

La separazione completa tra la macchina fisica e la macchina UML sembra poter risolvere il problema di garantire un server web facilmente accessibile e sicuro. Una eventuale intrusione nella macchina UML infatti non avrebbe effetto sulla macchina ospite, il che permette di allentare i vincoli sul server web rendendolo più facilmente fruibile, ed ottenere al contempo un livello di sicurezza più elevato. Il limitato accesso all'hardware da parte del kernel UML fa si che non si tratti semplicemente di uno spostamento del problema, ma piuttosto di una sua nuova definizione: il server web è ora completamente controllato dalla macchina ospite, che può essere una macchina blindata a piacere, senza che per questo venga ridotta la sua usabilità. Se il server web viene compromesso è molto più difficile per l'attaccante usare la macchina per i propri scopi, perché è strettamente controllata dall'host, o semplicemente renderla inutilizzabile, dato che come si vedrà nel seguito il ripristino del backup del server UML richiede solo pochi secondi. Naturalmente però non si deve pensare che sul server web UML si debbano trascurare le precauzioni di sicurezza basilari.

2 INSTALLAZIONE DEL SISTEMA

Si sono condotti i test di installazione di UML su un portatile di prova e su un server web di un esperimento.

Si è scelto di usare, sia per il sistema ospite che per UML, la distribuzione Gentoo-Linux [6], per le sue caratteristiche di flessibilità e configurabilità; le considerazioni fatte possono essere applicate però ad una generica distribuzione.

2.1 Macchina ospite

Non si daranno tutti i dettagli di installazione della Gentoo sulla macchina ospite, già ben documentati nel sito della distribuzione. Si vuole solo sottolineare che nella installazione sono state prese tutte le precauzioni di sicurezza delle quali si è parlato in precedenza.

Per quanto riguarda il kernel si è scelto di adottare la patch SKAS. Tradizionalmente UML funziona come segue:

- Ad ogni processo in UML corrisponde un processo sull'ospite
- Un thread speciale, il "tracing thread", intercetta le chiamate di sistema dei processi UML
- Il tracing thread annulla le chiamate di sistema, e le reindirizza nel kernel UML, che è accessibile direttamente dal processo UML

Questo tipo di funzionamento, chiamato "*modo TT*" ha dei problemi di sicurezza e velocità. Il kernel UML è presente nello spazio di indirizzamento di ognuno dei suoi processi, ed è scrivibile, almeno nella installazione standard. Questo è chiaramente un problema di sicurezza, dato che se è possibile scrivere nel kernel, che ricordiamo è un processo in user space nella macchina ospite, si apre la possibilità di accedere a tale macchina. Esiste una

modalità più sicura, detta “UML *jail*”, che rende i dati del kernel solo leggibili durante l’esecuzione di un processo, ma tale modalità è anche estremamente più lenta. Inoltre UML usa i segnali di sistema per controllare il suo kernel durante una chiamata di sistema o un interrupt, e questo meccanismo è intrinsecamente lento.

L’applicazione della patch SKAS (Separate Kernel Address Space) al kernel della macchina host fa si che il kernel UML venga eseguito in uno spazio di indirizzamento completamente differente da quello dei suoi processi, che quindi non possono più accedervi. Lo spazio di indirizzamento dei processi diviene identico a quello che avrebbero girando sulla macchina ospite e si hanno inoltre miglioramenti notevoli sul piano della velocità.

Per applicare la patch SKAS si è scaricato il kernel nella versione “vanilla”

```
host#emerge vanilla-sources
host#cd /usr/src/
host#wget http://prdownloads.sourceforge.net/user-mode-linux/host-skas3.patch
host#cd linux-2.4.24
host#cat ../host-skas3.patch | patch –p1
```

Se si usa il kernel 2.4.25 o un kernel della serie 2.6 si possono scaricare le relative patch all’indirizzo [7].

Durante la configurazione del kernel è necessario abilitare, con riferimento al menu disponibile con il comando `#make menuconfig`, le seguenti opzioni:

```
Code maturity level options --->
[*] Prompt for development and/or incomplete code/drivers"
General setup --->
[*]/proc/mm
File systems --->
<*> Ext3 journalling file system support
[*] Virtual memory file system support (former shm fs)
[*] /proc file system support
[*] /dev file system support (EXPERIMENTAL)
[*]  Automatically mount at boot
[] /dev/pts file system for Unix98 PTYs
<*> Second extended fs support
Networking options --->
      IP: Netfilter configuration --->
      <*>IP tables support
            <*>Full NAT
                  <*>MASQUERADE target support
Network device support --->
```

```
<*>Universal TUN/TAP device driver support
```

Si noti che le opzioni in rosso sono necessarie per il corretto funzionamento della distribuzione Gentoo 1.4. L'opzione riguardante */dev/pts* può essere deselezionata a meno che non si utilizzi un kernel 2.6. Naturalmente se si utilizzano altri filesystem andranno abilitate le corrispondenti opzioni nel menu di configurazione.

Un altro accorgimento da prendere nella configurazione dell'host riguarda il filesystem */tmp*. Come si è detto in precedenza la macchina UML non ha accesso diretto all'hardware della macchina ospite, e quindi neanche alla RAM. Per simulare la memoria utilizza dei file creati nel filesystem */tmp* della macchina ospite. Evidentemente questo rallenta notevolmente l'esecuzione dei programmi, in quanto ogni accesso alla memoria viene tradotto in un lento accesso su disco. Per migliorare la situazione è necessario modificare il file */etc/fstab* in modo tale che la directory */tmp* non sia montata su disco ma in memoria, aggiungendo la riga:

```
none        /tmp        tmpfs       defaults    0     0
```

In questo modo */tmp* verrà montata direttamente in RAM, sul filesystem **tmpfs**. Tale filesystem è simile ad un tradizionale disco RAM, con la differenza che alloca solo la memoria di cui ha effettivamente bisogno. La sua installazione standard prevede come dimensione massima la metà della memoria fisica disponibile, ma è possibile modificarne la grandezza, ad esempio a 512M, montandolo con l'opzione

```
size=512M
```

Tmpfs può essere però trasferito su disco (*swap*) se la memoria fisica non è sufficiente: questo rallenterebbe nuovamente il sistema, quindi è assolutamente necessario avere una macchina ospite con RAM sufficiente a coprire il fabbisogno del server UML.

Bisogna porre attenzione al fatto che se non si specifica nella riga di partenza del server UML l'opzione

```
mem=size
```

UML parte con soli 30 Mbyte di RAM. La dimensione della memoria allocata per UML può anche eccedere la memoria fisica della macchina, se /tmp è abbastanza grande, con il risultato di avere l'eccesso spostato nell'area di swap del disco e un generale rallentamento. Se si abilita nel kernel UML il supporto "highmem" il parametro size può salire fino a 4Gbyte, altrimenti è limitato a 512Mbyte.

Un confronto tra le prestazioni del sistema nei due casi si trova al capitolo 6.

2.2 Macchina UML

La distribuzione Gentoo fornisce i sorgenti di un kernel con la patch già applicata, quindi tutto quel che si deve fare è:

```
host#emerge sys-kernel/usermode-sources
host#emerge usermode-utilities
host#cd /usr/src/uml/linux
host#make menuconfig ARCH=um
host#make linux ARCH=um
host#cp linux /usr/local/bin/linux
```

E’ molto importante usare l’etichetta di architettura ARCH=um. Nella configurazione del kernel si deve porre attenzione alle seguenti opzioni:

```
Code maturity level options --->
[*] Prompt for development and/or incomplete code/drivers"
General setup --->
[*]Separate kernel address space support
[]Tracing Thread support
<>Host filesystem
<>Honeypot proc filesystem
[*]/proc/mm
Block devices --->
[*]Always do synchronous disk IO for UBD
File systems --->
<*> Ext3 journalling file system support
[*] Virtual memory file system support (former shm fs)
[*] /proc file system support
[*] /dev file system support (EXPERIMENTAL)
[]  Automatically mount at boot
```

Si è disabilitato il supporto al tracing thread, per evitare ulteriori penalizzazioni della velocità e per le questioni di sicurezza discusse in precedenza. Sempre per aumentare la sicurezza si è negata la possibilità di raggiungere il filesystem dell’host da parte del sistema UML, e la possibilità da parte dell’host di sovrascrivere la directory */proc* di UML; si tratta di opzioni utili in altri scenari. Si faccia molta attenzione ad includere nel kernel il supporto al filesystem con il quale si è formattato il file *root_fs*, ovviamente se non si riesce a leggere tale file non è possibile neanche caricare dei moduli.

2.2.1 Creazione del filesystem

Il kernel UML ha un modulo, conosciuto come “UML block drive”, che permette di far corrispondere ad un filesystem dell’ospite un *device ubd*, e quindi di renderlo disponibile per il sistema UML. Normalmente un block device di questo tipo è associato ad un file sull’host che contiene il filesystem. Questo file all’interno della macchina UML viene visto come un normale block device, e il filesystem in esso viene montato come accade ad un normale filesystem su un disco fisico. Se il filesystem del block device è del tipo *swap*, il kernel lo userà come una partizione di swap: infatti lo scheduler dell’host semplicemente implementa le decisioni prese dallo scheduler UML che viene eseguito indipendentemente. Similmente UML ha la propria VM, che alloca la memoria da quella che lui considera memoria fisica, rendendola disponibile come memoria virtuale dei processi che la richiedono. E’ chiaro quindi che se esiste la possibilità di usare memoria di swap, questa viene utilizzata.

Per creare il filesystem di root è innanzitutto necessario predisporre un file:

```
host# dd if=/dev/zero of=root_fs seek=1000 count=1 bs=1M
```

Questo comando crea uno “sparse file” di nome *root_fs*, con la caratteristica di avere dimensione di 1 GB, ma spazio effettivamente allocato pari inizialmente a 1MB, come si può verificare con i comandi:

```
host$df –h
host$ls –lh.
```

Quando si scrive su un file di questo genere lo spazio allocato aumenta di conseguenza, con un massimo pari alla dimensione del file.

Sarà utile anche predisporre con analoghi comandi un file di swap e un file sul quale montare la directory degli utenti, separata dal sistema vero e proprio.

```
host#dd if=/dev/zero of=swap_fs seek=500 count=1 bs=1M
host#dd if=/dev/zero of=home_fs seek=500 count=1 bs=1M
```

Sui file viene poi creato il filesystem ext3 e quello di swap

```
host#mke2fs –Fj root_fs
host#mke2fs –Fj home_fs
host#mkswap –f swap_fs
```

A questo punto è necessario realizzare la struttura della distribuzione Gentoo, in modo da poterla portare sul file di root. Per semplicità si è scelto di partire dallo stage3 della distribuzione. Il comando

```
host#links http://gentoo.oregonstate.edu/releases/x86
```

permette di scaricare il relativo file *.tar.bz2*. Ottenuto il file lo si estrae in una directory apposita:

```
host#mkdir /mnt/gentoo
host#mv stage3-arch-version.tar.bz2 /mnt/gentoo
host#cd /mnt/gentoo
host#tar -xvjpf stage3-arch-version.tar.bz2
```

Per effettuare le necessarie personalizzazioni e gli aggiornamenti della distribuzione è necessario portarsi con chroot nella struttura di directory appena creata:

```
host#cp /etc/resolv.conf  /mnt/gentoo/etc
host#mount –o bind /proc /mnt/gentoo/proc
host#chroot /mnt/gentoo /bin/bash
uml#ln -sf /usr/share/zoneinfo/path/to/timezonefile /etc/localtime
```

Ora è possibile, se si vuole, procedere all'aggiornamento della distribuzione:

```
uml#emerge sync
uml#env-update
uml#source /etc/profile
uml#cp /etc/make.conf /etc/make.conf.bak
uml#CONFIG_PROTECT="-*" emerge –u world
uml#cp /etc/make.conf.bak /etc/make.conf
```

Come per l'installazione di un server Gentoo standard vanno aggiunti i servizi fondamentali *cron* e *sysklogd*:

```
uml#emerge <nome-servizio>
uml#rc-update add <nome-servizio> default
uml#crontab /etc/crontab
```

Per far si che il sistema monti correttamente i block device bisogna modificare il file */etc/fstab* come segue:

```
/dev/ubd/0          /               ext3        noatime     0 0
/dev/ubd/1          none            swap        sw          0 0
/dev/ubd/2          /home           ext3        noatime     0 0
```

Il numero di device e quindi di file del sistema ospite da utilizzare può variare a seconda delle esigenze. Dato che il sistema *uml* alla fine della configurazione deve funzionare da web server è necessario installare apache.

```
uml#emerge apache
```

A questo punto dopo aver impostato la password di root e il nome dell'host si può uscire dalla chroot, smontare le directory e comprimere la distribuzione in un unico file, che può essere utile per ripartire da zero:

```
uml#passwd
uml#echo mymachine > /etc/hostname
uml#exit
host#umount /mnt/gentoo/proc
host#cd /mnt/gentoo
host#du –hs /mnt/gentoo
host#tar -cvjpf ~/gentoo.tbz2 *
host#rm /mnt/gentoo
```

Il comando *du* può servire per calibrare più precisamente la dimensione del file di root. L'unica cosa che rimane da fare è trasferire la distribuzione su tale file:

```
host#mount –o loop root_fs /mnt/loop
host#tar xvjpf gentoo.tbz2 –C /mnt/loop
host#umount /mnt/loop
```

Si può ora far partire la macchina *uml* con il comando:

```
host$linux ubd0=root_fs ubd1=swap_fs ubd2=data_fs
```

dove si noti che l'utente deve essere per sicurezza un account non privilegiato, creato per l'occasione, con i minimi privilegi possibili compatibili con l'utilizzo di UML.

Con la configurazione standard del kernel però questo comando deve essere eseguito in ambiente grafico, perché UML usa xterm per aprire le console virtuali al boot; deve quindi essere impostata propriamente la variabile DISPLAY e deve essere possibile accedere al server X. Naturalmente non è affatto detto che si voglia avere un terminale grafico sul server web, così come non è detto che servano le sei console virtuali che vengono aperte per default. Per ovviare al problema da un terminale testuale si può usare il comando:

```
host$linux ubd0=root_fs ubd1=swap_fs ubd2=data_fs con=null con0=fd:0,fd:1
```

```
con1=tty:/dev/tty7
```

In questo modo la console principale *con0* viene inviata allo stdin e stdout, la console 1 viene inviata su una console non usata dalla macchina ospite e può essere usata per il login su UML dalla console dell'host, le altre console semplicemente non vengono create.

2.2.2 Backup del filesystem

Dato che il filesystem è contenuto in due file della macchina ospite la prima cosa che viene in mente per realizzare una copia di sicurezza è di copiare i file periodicamente in un posto sicuro. Ma se durante la copia il server UML cambia il contenuto del suo filesystem scrivendo, si potrà avere nel backup una situazione non consistente. Per ovviare a questo problema si possono usare i comandi di stop e go delle usermode utilities

Nel kernel UML deve essere abilitata l'opzione seguente:

```
General setup --->
      [*]Management console
      [*]Magic SysRq key
```

Al comando di boot della macchina virtuale è comodo aggiungere un nome:

```
umid=web_server
```

Al boot verrà così creato il file */home/user/.uml/web_server/console*, che è il socket che la console di gestione userà per comunicare con il server. I comandi per il backup saranno quindi:

```
host#uml_mconsole web_server stop
host#uml_mconsole web_server sysrq s
host#cp -fra --sparse=always root_fs /rootfs_bak
host#uml_mconsole web_server go
host#nice 19 bzip2 rootfs_bak
```

In questo modo si ferma la macchina *uml*, si sincronizzano tutti i filesystem, si copia il file in uno di backup, si fa ripartire la macchina ed infine si comprime il file. L'operazione non comporta il reboot del server, ma comunque un suo temporaneo stop dell'ordine del minuto, e comunque dipendente dalle dimensioni del file di root.

3 CONFIGURAZIONE DELLA RETE

Nella precedente descrizione manca la configurazione della connessione di rete della macchina UML, indispensabile per permetterle di operare da web server. La macchina host si comporta da firewall, essendo l'unico punto di contatto tra la rete privata costituita dalla macchina UML e la rete pubblica alla quale è collegata. E' importante che le opzioni relative alla rete nel kernel dell'host siano state correttamente configurate, così come indicato nel paragrafo 2.1.

Il primo passo consiste nel dotare la macchina UML di una interfaccia di rete, descrivendola nella riga di comando. Ad esempio, ma non è l'unico modo possibile di operare, una interfaccia ethernet della macchina UML può essere connessa ad una interfaccia TUN/TAP dell'host: il device TAP può essere descritto come un device ethernet che invece di ricevere pacchetti ed inviarli su un supporto fisico li riceve e li invia ad un programma in user space. La stessa cosa vale per il device TUN, che opera a livello IP. Il comando per impostare l'interfaccia eth0 dalla riga di comando è:

```
eth0=tuntap,,,192.168.0.254
```

Questo comando imposta una interfaccia eth0 all'interno della macchina UML, le assegna un indirizzo ethernet MAC dipendente dall'indirizzo IP che le verrà assegnato in seguito, connettendola al primo device tap dell'ospite, sul quale imposta un indirizzo IP. Il comando non è però sicuro, in quanto utilizza un eseguibile, *uml_net*, per configurare al volo il device tap della parte host della rete privata: se un utente riesce all'interno della macchina UML a cambiare l'indirizzo IP della interfaccia, tale programma cambierà di conseguenza le tabelle di routing dell'host. Ciò permetterebbe di poter inviare alla macchina UML ad esempio tutti i pacchetti destinati ad una terza macchina da attaccare. Il problema può essere risolto configurando manualmente un device tap sulla macchina ospite, come si farà nel seguito.

L'indirizzo IP assegnato alla parte ospite del tap device deve essere diverso dall'indirizzo IP assegnato al device ethernet UML; è possibile utilizzare per il device tap l'indirizzo IP della interfaccia eth0 dell'ospite, internamente però UML deve avere un indirizzo IP unico per la propria interfaccia di rete.

3.1 Configurazione con ip privato

Supponiamo di voler assegnare l'indirizzo privato 192.168.0.1 alla macchina UML, e l'indirizzo 192.168.0.254 al device *tap0* della macchina ospite, in modo da ottenere un server virtuale che risponde alle richieste web indirizzate alla macchina reale.

Nella macchina ospite deve essere abilitato l'IP Masquerading:

```
host#iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to ip.pubblico.server.host
```

Il Masquerade è un particolare caso di NAT (Network Address Translation): la regola precedente dice al firewall che tutti i pacchetti che stanno per essere inviati sulla interfaccia eth0, e quelli futuri che apparterranno alla connessione, devono essere modificati in modo che il loro indirizzo IP di provenienza sia quello del server host, anche se provengono dal server UML.

E' necessario abilitare la proprietà del kernel di reindirizzare i pacchetti tra le varie interfacce ethernet di cui dispone, in quanto servirà per la connessione tra la rete privata host - UML e l'interfaccia eth0:

```
host#echo 1 > /proc/sys/net/ipv4/ip_forward
```

Si deve ora creare un device TAP appartenente all'utente *uid* dal quale UML verrà eseguito, e configurarlo con l'indirizzo di rete:

```
host#tunctl –u uid
host#ifconfig tap0 192.168.0.254 netmask 255.255.255.255 up
```

Alla tabella di routing deve essere aggiunto l'indirizzo che verrà poi assegnato alla interfaccia di rete della macchina UML:

```
host#route add -host 192.168.0.1 dev tap0
```

Per permettere agli altri host della rete di vedere la macchina UML è necessario abilitare il proxy arp sul device tap0:

```
host#echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp
host#arp -Ds 192.168.0.1 eth0 pub
```

In maniera sintetica si può dire che il proxy arp é una funzionalità che permette ad un router linux di rispondere ad una richiesta ARP, che "vede passare" su un'interfaccia, anche quando il destinatario è su un'altra interfaccia del router stesso. Nel caso in esame il router è la macchina host, che risponderà alle richieste ARP indirizzate alla macchina *uml* sulla interfaccia eth0.

Il device */dev/net/tun* deve essere scrivibile dall'utente che usa UML. Nel modo più semplice:

```
host#chmod 666 /dev/net/tun
```

In questo modo però chiunque può creare device tap, il che non è una buona cosa per motivi di sicurezza (possibili attacchi DoS). Una alternativa più sicura consiste nel creare un gruppo contenente gli utenti con device tap preconfigurati, e usare *chgrp* su */dev/net/tun* dando

permessi 660:

```
host#groupadd uml
host#usermod -Guml user-uid
host#chgrp uml /dev/net/tun
host#chmod 660 /dev/net/tun
```

Ultimata questa preparazione dell'host, che naturalmente affinché sia recuperata al riavvio andrà salvata in un file che verrà eseguito da */etc/conf.d/local.start* (Appendice C), è possibile eseguire UML dall'utente *user* con il comando

```
host$linux ubd0=root_fs ubd1=swap_fs ubd2=home_fs con0=fd:0,fd:1 con1=tty:/dev/tty7 con=
null eth0=tuntap,tap0 mem=512M
```

Dalla console 7 dell'host si potrà entrare nella macchina UML come root. Ora si può far partire l'interfaccia di rete:

```
uml#ifconfig eth0 192.168.0.1 up
```

Naturalmente anche questa impostazione, come altre nel seguito, può essere inclusa in uno file di script da far lanciare da */etc/conf.d/local.start*, questa volta nel server UML.

A questo punto la interfaccia di rete *tap0* dell'host dovrebbe essere raggiungibile, e lo si può verificare con

```
uml#ping 192.168.0.254
```

Se così non è si controlli il firewall iptables dell'host e si disabiliti la tabella *filter*, della quale ci si occuperà in seguito, con il comando:

```
host#iptables -F
```

Non rimane che aggiungere il default gateway e la configurazione del DNS:

```
uml#route add default gw 192.168.0.254
uml#scp user@192.168.0.254:/etc/resolv.conf /etc/resolv.conf
```

La rete configurata in questo modo permette le connessioni a partire dalla macchina UML, ad esempio è possibile usare il comando *emerge sync*, ma poiché non si utilizza un indirizzo pubblico il server non è raggiungibile dall'esterno.

Se si vuole ottenere come nei progetti una macchina virtuale che funzioni da web server,

rispondendo sullo stesso indirizzo pubblico della macchina fisica bisogna abilitare il *port-forwarding*:

```
host#iptables -t nat -A PREROUTING -p tcp -i eth0 -d ip.pubblico.server.host --dport 80 -j DNAT --to 192.168.0.1:80
host#iptables -A FORWARD -p tcp -i eth0 -d 192.168.0.1 --dport 80 -j ACCEPT
```

In questa maniera ogni richiesta web indirizzata al server host verrà girata al server virtuale UML, che risponderà al suo posto. Analogamente si può procedere per la porta ssh:

```
host#iptables -t nat -A PREROUTING -p tcp -i eth0 -d ip.pubblico.server.host --dport 22 -j DNAT --to 192.168.0.1:22
host#iptables -A FORWARD -p tcp -i eth0 -d 192.168.0.1 --dport 22 -j ACCEPT
```

3.1.1 Configurazione di iptables

Oltre alle regole di NAT sopra descritte è utile impostare sulla macchina host un completo set di regole di firewall, che proteggano la macchina UML e lo stesso host da attacchi esterni. [8] [9] Per semplicità nella appendice C si riporta un esempio di configurazione finale in forma di shell script, che è possibile far eseguire modificando il file */etc/conf.d/local.start*. Un altro modo di utilizzo consiste nell'eseguire lo script una volta, poi salvarlo con

```
host#iptables-save > /var/lib/iptables/rules-save
host#rc-update add iptables default
```

Una premessa importante per poter leggere correttamente il file: ogni volta che un pacchetto transita per una catena di iptables tutte le regole vengono controllate in ordine, dalla prima inserita fino all'ultima. Se una regola risulta applicabile allora al pacchetto viene applicata l'azione specificata e viene interrotto il processo di ricerca. Se non vengono trovate regole opportune la sorte del pacchetto è scelta in base alla policy della catena.

Le regole proposte, commentate nel file di configurazione hanno le seguenti caratteristiche fondamentali:

1. Permettono la connessione web ed ssh al server UML utilizzando l'indirizzo dell'host.
2. Bloccano ogni utilizzo della rete a partire dalla macchina UML.
3. Solo un host autorizzato può usare il ping per controllarne la presenza in rete del web server

Si noti che pur fornendo agli utenti un accesso interattivo al server web tramite ssh, non è possibile che un utente malintenzionato, anche con i privilegi di root, usi la macchina per attacchi verso l'esterno, in quanto l'unica attività di rete consentita è la risposta a connessioni

esterne di tipo web ed ssh. Inoltre non è possibile modificare il firewall della macchina host, che inaccessibile per gli utenti del server web.

Nel caso in cui si voglia accedere direttamente alla rete dalla macchina UML basta cambiare la policy di FORWARD nell'host. Nel modo più semplice:

```
host#iptables -P FORWARD ACCEPT
```

3.2 Configurazione con ip pubblico

Se è necessario avere un indirizzo pubblico per il server web diverso da quello della macchina ospite si procede variando leggermente la procedura descritta.

Sull'host si può assegnare, per non sprecare indirizzi, al device *tap0* l'indirizzo della interfaccia di rete *eth0*:

```
host#tunctl –u uid
host#ifconfig tap0 ip.pubblico.server.host netmask 255.255.255.0 up
```

All'interno della macchina UML la configurazione della rete viene realizzata con i seguenti comandi:

```
uml#ifconfig eth0 ip.pubblico.server.uml
uml#route add ip.pubblico.server.host eth0
uml#route add default gw ip.pubblico.server.host
```

Infine deve essere modificato il firewall: si elimina completamente la parte relativa al forward delle richieste Web e ssh, e si modifica la parte della catena di FORWARD a seconda delle proprie esigenze, controllando così tutto il traffico in entrata ed uscita dal server web. Ad esempio i seguenti comandi, più permissivi di quelli indicati in precedenza, permettono di accedere al server uml tramite web e ssh, e di uscire dal server sulla rete tramite ssh, emerge (rsync) e web:

```
host#iptables –P FORWARD DROP
host#iptables –A FORWARD –m state --state RELATED, ESTABLISHED -j ACCEPT
host#iptables –A FORWARD -p tcp -m tcp --dport 80 -j ACCEPT
host#iptables –P FORWARD -p tcp -m tcp --dport 22 -j ACCEPT
host#iptables –P FORWARD -p udp -m udp -i tap0 --dport 53 -j ACCEPT
host#iptables –P FORWARD -p udp -m udp -i tap0 --dport 873 -j ACCEPT
```

4 TRASFERIMENTO DI UN SERVER PREESISTENTE

I vantaggi di UML possono essere sfruttati anche se si possiede già un server web funzionante, trasferendo le funzioni dal server reale a quello virtuale al prezzo di uno stop della macchina.

Innanzitutto deve essere ricompilato il kernel host con la patch SKAS, ed l'eseguibile corrispondente al kernel UML: le indicazioni da seguire sono le stesse date in precedenza nel paragrafo 2.1 e 2.2.

Poi si passa alla creazione dei file che ospiteranno il filesystem. Se si vuole riprodurre fedelmente la struttura del server originario si deve creare un file per ogni partizione montata dall'host.

A questo punto è necessario lo stop delle attività dell'host, sia per il caricamento del nuovo kernel, sia per "congelare" lo stato del server web. Per copiare il filesystem sui block device UML è possibile usare il device di loop, che deve essere quindi abilitato nel kernel host. Ad esempio per la partizione / di root si ha:

```
host#mkdir /loop
host#mount -o loop root_fs /loop
host#cp -avx / /loop
host#cp -avx /dev /loop
host#umount /loop
```

In questo modo la copia riprodurrà fedelmente il filesystem, e si limiterà alla sola partizione sulla quale è montata la directory principale. Si faccia molta attenzione al fatto che il filesystem */dev* è differente da quello principale, e quindi deve essere copiato a parte. La stessa cosa può essere fatta con le altre partizioni presenti nell'host.

Affinché la macchina UML possa essere accessibile dalla rete è necessario configurare sia la parte host sia la parte UML con le stesse modalità descritte al paragrafo 3. Per la configurazione della macchina virtuale si usa sempre il device di loop:

```
host#mount -o loop root_fs /loop
host#chroot /loop /bin/bash
```

Una volta avuto accesso al file system è necessario modificare il file */loop/etc/fstab* in modo che il sistema monti correttamente i block device, come indicato al paragrafo 2.2.1. Il servizio infine può riprendere esattamente da dove è stato interrotto: per gli utenti la migrazione sarà trasparente, se si ha cura di usare per il web server lo stesso indirizzo che aveva in precedenza.

Uno degli aspetti che potrebbe indurre confusione nell'utente è il fatto che i filesystem appaiono montati sotto gli strani device */dev/ubd/0*. Se si fa partire UML con l'opzione *fakehd* verranno creati i device */dev/hd/disc0/disc*, molto più somiglianti ad un disco fisico, sui quali poter montare i filesystem modificando il file */etc/fstab*.

5 TEST DELLE PRESTAZIONI

La prima cosa che si è verificata è stata la effettiva differenza di prestazioni nei due casi in cui il filesystem */tmp* dell'ospite sia montato su disco o in memoria RAM.

Come applicazione di benchmark si è usata la compilazione del kernel 2.4.25 all'interno della macchina UML, su un server di prova dotato di Xeon a 2Ghz, Controller Adaptec 2400A RAID 5, 1GB di RAM, e kernel vanilla 2.4.25 con la patch SKAS. La compilazione del kernel linux è un buon esempio di applicazione reale, perché richiede sia input-output dai dischi sia elaborazione dalla CPU. I risultati sono stati i seguenti:

TAB. 1: Tempi di compilazione

	ext3 2.4.24r1	**tmpfs 2.4.24r1**	**PC originario**
Real	18m42	7m38	3m50
User	12m0	5m5	3m32
Sys	5m43	2m31	0m17

Nella intestazione di ogni colonna è riportato il filesystem sul quale è montato */tmp* e il kernel UML usato. Anche se l'uso del filesystem tmpfs aumenta la velocità, il sistema si rivela comunque più lento rispetto alla macchina originaria di circa un fattore 2.

Per cercare di isolare il contributo al rallentamento dell'attività di lettura e scrittura sul filesystem da quello dovuto alla esecuzione del kernel nello user space dell'ospite, ovviamente nella situazione più favorevole in cui */tmp* è montata su tmpfs, si sono usati sul server di prova dei programmi di benchmark. Per il test del filsystem si è utilizzato IOzone Filesystem Benchmark [10]: il programma è stato ricompilato sulla macchina e lanciato con il comando

```
host#iozone -Razb output.wks –g 2G -S 512 –i0 –i1 -i2
```

In questo modo si è usata una dimensione massima del file pari a circa due volte la RAM per evitare gli effetti del buffering del filesystem in RAM, e si sono effettuati i test di scrittura/riscrittura, lettura/rilettura e di lettura e scrittura casuale. In ambedue i casi il filesystem utilizzato è stato ext3. Un esempio dei risultati ottenuti si può esaminare nelle figure 1 e 2, dove sono riportati i dati relativi alla lettura casuale rispettivamente su host e su server UML:

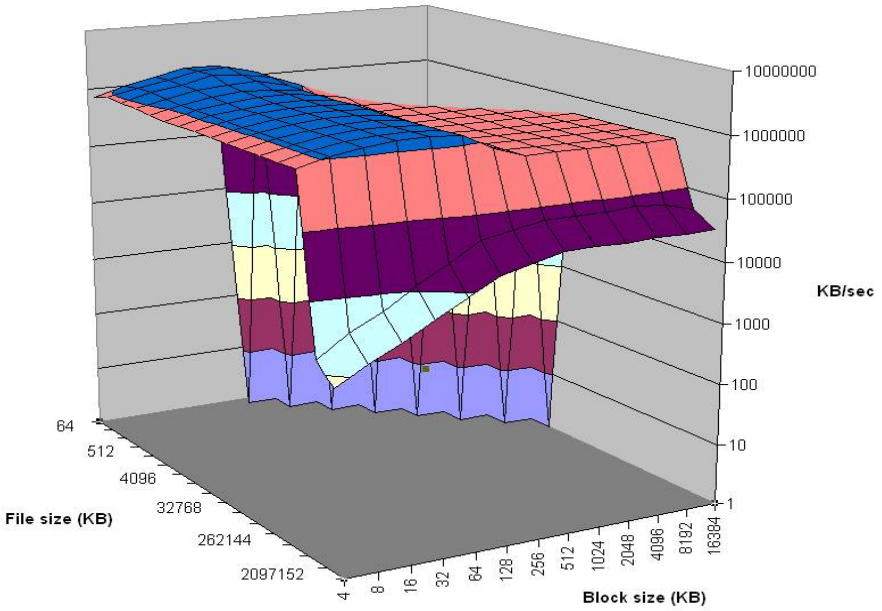

FIG 1: Velocità di lettura casuale host

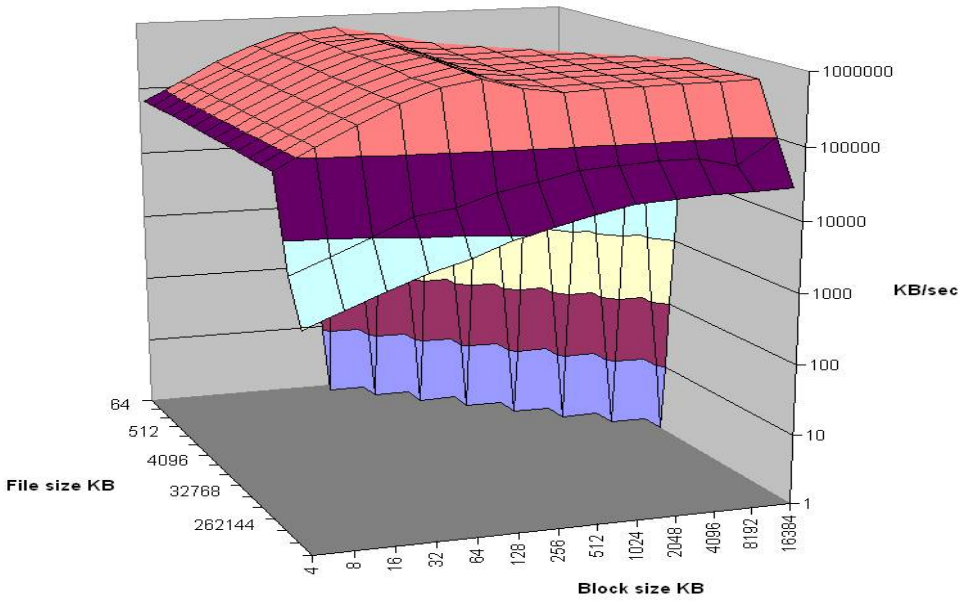

FIG 2: Velocità di lettura casuale UML

Si nota nel grafico relativo all'host la diminuzione della velocità all'aumentare della dimensione dei blocchi di scrittura per effetto della cache del processore, pari a 512KB. La brusca diminuzione che si ha per dimensione del file maggiore di 512MB è dovuta invece alla azione del buffer del filesystem in RAM. In sintesi la velocità del filesystem, al netto degli effetti di cache del disco e della CPU, è data dalla serie corrispondente al file di 2 GB, per blocchi abbastanza grandi da non risentire della velocità fisica delle testine. Gli stessi effetti si possono vedere nel grafico relativo al server *uml*.

E' interessante notare che, sebbene la velocità di lettura casuale fuori cache sia confrontabile nei due casi, dando valori di 29077 KB/sec per l'host e di 25128 KB/sec per l'UML, nell'uso normale del filesystem gli effetti di cache si fanno molto marcati. Ad esempio nella figura 3 è riportata la velocità di scrittura casuale nel caso di block size 4KB, corrispondente al default block size per ext3, e 64KB, che è la dimensione degli stripe block dell'array IDE:

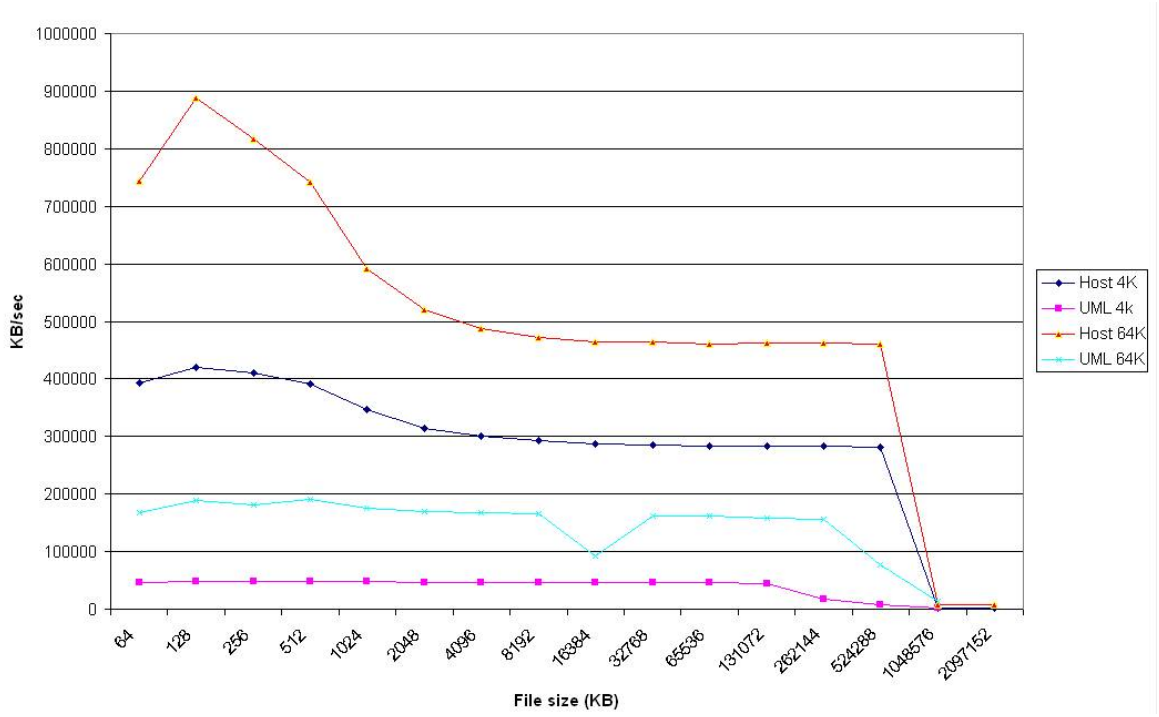

FIG 3: Confronto velocità di scrittura casuale

Come si vede UML è più lento nelle operazioni sul filesystem se si assumono per le dimensioni dei file e le dimensioni dei blocchi di scrittura i valori più comunemente utilizzati. Una media delle velocità di scrittura casuale e di lettura casuale fatta sulle dimensioni dei file e dei blocchi da come risultato il seguente:

TAB. 2: Velocità media filesystem KB/sec

	UML	**PC host**
Write	87685	119621
Read	428162	748149
Random write	127485	370004
Random read	389897	776474

I risultati sono coerenti con il comportamento del RAID 5 hardware, che è più lento in scrittura che in lettura perché deve calcolare la parità dei dati, e con la procedura usata dal programma di test per le misure: prima viene scritto il file di prova, e quindi la scrittura sequenziale è l'operazione più lenta, e poi su quello stesso file vengono eseguiti gli altri test, che in gran parte fanno uso della cache del processore e del buffer del filesystem.

Il confronto mostra un decadimento medio delle prestazioni del filesystem sotto UML di circa un fattore 1,8.

Il test della velocità della memoria è stato effettuato con il semplice benchmark MemSpeed [11], che scrive e legge in memoria blocchi di grandezza crescente fino a raggiungere i 512Mbyte, e con Nbench-byte [12]:

```
host/uml#./MemSpeed (16 512)
host/uml#./nbench
```

I risultati si possono leggere nella tabella 3 e nelle figure 4 e 5:

TAB. 3: Indici nbench

	UML	**PC host**
Memory	9606	9687
Integer	6455	6597
Floating point	12980	13077

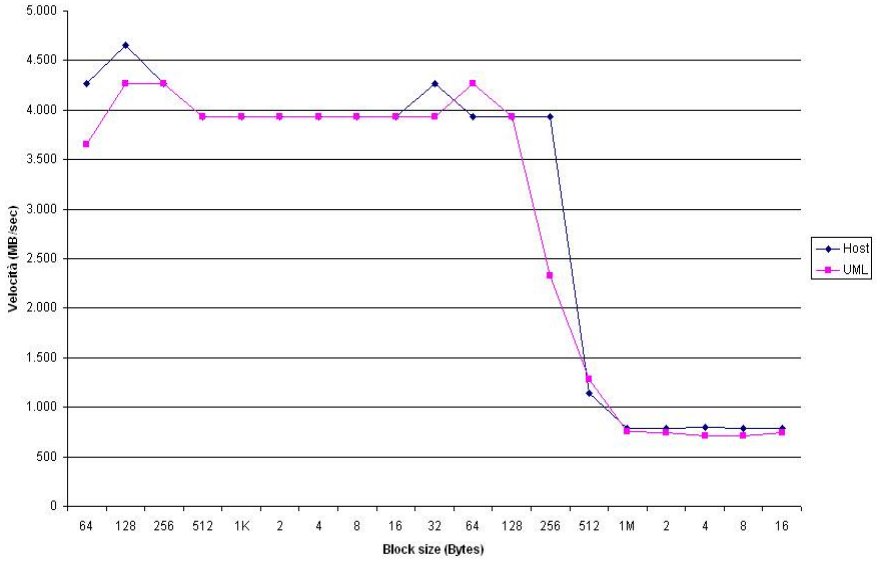

FIG 4: Velocità di scrittura su RAM

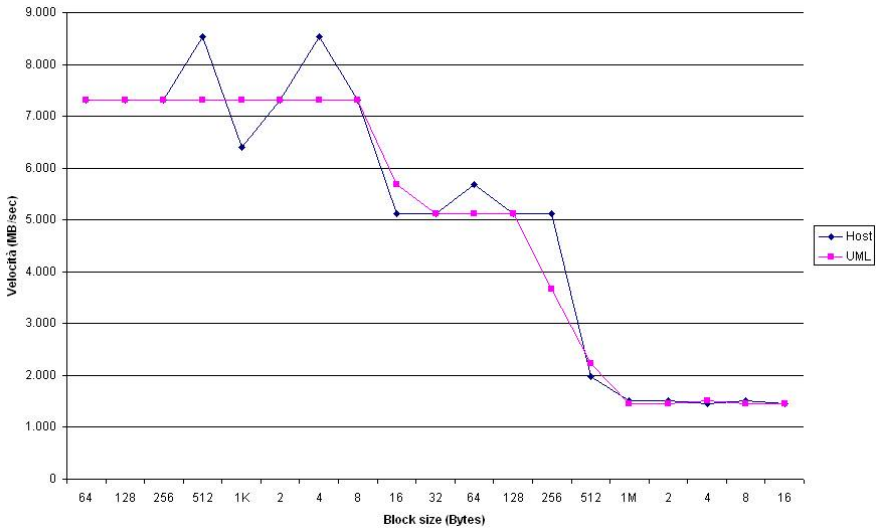

FIG 5: Velocità di lettura su RAM

Nei due test è evidente l'effetto della cache L2 del processore: se i blocchi superano i 512 KB le prestazioni decadono velocemente. Nel test di lettura si nota anche il gradino corrispondente alla cache L1 di 8KB.

I semplici test condotti mostrano che il rallentamento è dovuto quasi esclusivamente all'accesso al filesystem. A questo proposito si fa notare che la tecnica di assegnare al server UML un filesystem contenuto in un singolo file dell'host non è la sola percorribile, perché ad

un block device UML può essere anche associata una generica partizione. Non si sono qui approfonditi però i problemi di sicurezza che tale soluzione potrebbe presentare.

5 CONCLUSIONI

Nel presente documento si è illustrata nel dettaglio la tecnica per creare all'interno di un server Linux un server virtuale, completamente isolato dal server principale, ed accessibile dalla rete esterna. Questa possibilità nel caso in esame è stata considerata per le implicazioni sulla sicurezza e la facilità di gestione di un server web, sebbene sia chiaro come questa, come nessuna altra singola misura, sia capace da sola di garantire completamente la sicurezza di una macchina.

Le misure condotte hanno dimostrato un calo di prestazioni del server virtuale, dovuto alla lentezza con la quale accede al suo filesystem. La sua accettabilità rimane da valutare caso per caso, e probabilmente ne esclude l'uso per server che abbiano necessità di intenso e rapido input-output.

6 RIFERIMENTI E BIBLIOGRAFIA

[1] Day Kevin, *Security Jungle, Le decisioni giuste per la sicurezza aziendale*, Mondadori.
[2] Logcheck http://sourceforge.net/projects/logcheck/
[3] Tripwire http://www.tripwire.org/
[4] Iptables http://www.netfilter.org/
[5] UML http://user-mode-linux.sourceforge.net/
[6] Gentoo http://www.gentoo.org/
[7] SKAS Patch http://www.user-mode-linux.org/~blaisorblade/
[8] IPTables for fun http://www.commedia.it/ccontavalli/index.html
[9] LinWiz firewall generator http://www.lowth.com/LinWiz/1.09/
[10] IOZone http://www.iozone.org/
[11] MemSpeed http://home.tvd.be/cr26864/Linux/PPC/MemSpeed.html
[12] nbench http://www.tux.org/~mayer/linux/bmark.html
[13] RSSH http://sourceforge.net/projects/rssh/

APPENDICE A – LE REGOLE FONDAMENTALI DELLA SICUREZZA

1-Regola del minimo privilegio

Consentire solo l'accesso necessario per eseguire il lavoro e niente di più. Inoltre consentire solo l'accesso del quale un individuo, gruppo o soggetto può essere responsabile in modo sicuro.

In altre parole: negare in partenza ogni privilegio agli utenti e consentire solo ciò che è strettamente necessario, piuttosto che garantire tutto e proibire quanto non desiderato.

2-Regola del cambiamento

Ogni cambiamento deve essere gestito e coordinato; se ne devono valutare le implicazioni sulla sicurezza e deve essere implementato solo dopo che se ne sia dimostrata la sicurezza. Naturalmente seguendo la regola precedente nessuno deve essere autorizzato ad apportare cambiamenti se non è qualificato per farli.

3-Regola della fiducia

Quando un'organizzazione si fida di qualcuno o qualcosa, affronta un certo grado di rischio. Fidarsi di qualunque soggetto significa che ci si sta fidando anche di tutti e di tutto ciò che ha accesso a tale soggetto, stabilendo una catena di fiducia. E' quindi necessario sempre sapere a chi si sta concedendo fiducia, e quali sono i rischi correlati.

4-Regola dell'anello debole

Una procedura di sicurezza è forte soltanto quanto il suo controllo più debole. Le fonti più comuni di "anello debole" sono le seguenti:

- Installazioni di default
- Password scadenti di utenti ed amministratori
- Modem attivi connessi a desktop, server e router
- Negligenza nel controllo dei log e degli strumenti di monitoraggio
- Connessioni di backup o ridondanti non protette
- Server, workstation ed altri dispositivi implementati per uso temporaneo
- Backup trascurati e non testati
- Applicazioni non autorizzate
- Software antivirus obsoleti

5-Regola della separazione

Per mantenere un alto livello di protezione, è importante separare gli oggetti in diversi livelli di sicurezza e applicare a essi diverse regole di accesso. Più compiti un oggetto deve svolgere, più problemi è verosimile che abbia, così come più sono i soggetti che hanno accesso ad un oggetto, più alta è la possibilità che questo sia esposto ad un rischio.

6-Regola della procedura a tre passi

Bisogna pensare a tutte le misure di sicurezza come ad un processo a tre fasi: implementazione, monitoraggio, manutenzione. Se manca una qualunque delle tre fasi, la sicurezza risultante sarà minima, se mai ce ne sarà.

7-Regola della azione preventiva (sicurezza proattiva)

Non si può lasciare che siano solo delle policy e procedure a posteriori a guidare le risposte alla sicurezza. L’obiettivo principale non è liberarsi da un attacco, ma impedire che esso possa mai avvenire. Un’organizzazione deve essere concentrata sulla sicurezza prima che se ne manifesti la necessità, non dopo.

Per questo deve essere routine quotidiana controllare l’esistenza di nuove vulnerabilità ed exploit, applicare patch e mantenersi al corrente dei problemi della sicurezza.

8-Regola della risposta immediata ed adeguata

Che un attacco abbia successo o meno deve scattare comunque una azione organizzata per investigarne i dettagli, analizzare i rischi esistenti e pianificare i passi futuri. Ogni organizzazione deve avere un piano organizzato per reagire durante un incidente. Questo piano deve essere chiaro, conciso e regolarmente aggiornato. Tutti devono avere dimestichezza con la parte di esso che li riguarda.

APPENDICE B – SICUREZZA DI BASE IN SISTEMI MULTIUTENTE

1. Disabilitare i SUID superflui. Per individuare tutti i file con SUID attivo eseguire:
   # `find / -type f \( -perm -04000 -o -perm -02000 \) -exec ls -l {} \;`
   Per disabilitare il SUID:
   # `chmod a-s <nome eseguibile>:`

2. Disabilitare tutti i protocolli di comunicazione in chiaro (rcp, rlogin, rsh, telnet, ftp) con il comando:
   # `chmod 0 <nome eseguibile>`
   Nessun server in chiaro, come telnet-server ad esempio, deve essere presente sulla macchina.

3. Il controllo diretto sulle directory */home* e */tmp* permette di proibire al loro interno file con il bit suid attivo ed eseguibili modificando il file */etc/fstab* come segue:
   ```
   /dev/sda#       /home       ext3        defaults,nosuid              1 2
   /dev/sda#       /tmp        ext3        defaults,nosuid,noexec       0 0
   ```
   In questo modo si vieta l'esistenza di eseguibili nell'area temporanea e l'attivazione nell'area utente di programmi suid.

4. Per le comunicazioni usare solo ssh e sftp, e solo con il protocollo 2: nel file */etc/ssh/sshd_config e /etc/ssh/ssh_config* eliminare il commento davanti alla riga Protocol 2, 1 e modificarla in
   ```
   Protocol 2
   ```
   Di norma è anche una buona idea impedire il login di root eliminando nel file */etc/ssh/sshd_config* il commento da:
   ```
   PermitRootLogin no
   ```

5. Collocare il server fisicamente al sicuro in un luogo protetto. Se ciò non è possibile cercare di cautelarsi contro la possibilità che qualcuno possa prenderne il controllo con un riavvio della macchina. Quindi:
   - Nel BIOS del PC impostare come primo device di boot l'hard disk: in questo modo non è possibile far partire la macchina con un CD o un floppy
   - Impostare una password per l'accesso al BIOS
   - Con i bootloader LILO e GRUB è possibile impedire ad un utente non autorizzato di cambiare le impostazioni del kernel. Per farlo bisogna aggiungere in */etc/lilo.conf*, prima della definizione delle immagini, le righe:
     ```
     restricted
     password=<password>
     ```
     oppure in */boot/grub/grub.conf* la riga
     ```
     password = <password>
     ```
   - Evitare infine che i due file di configurazione siano leggibili e modificabili da

tutti gli utenti:

```
# chmod 600 /etc/lilo.conf
```

6. Limitare forzatamente il periodo di validità delle password utenti, modificando il file */etc/login.defs:*

```
PASS_MAX_DAYS 180
PASS_WARN_AGE 14
```

7. Limitare la quantità di risorse disponibili al singolo utente, in modo da rendere più difficoltosi gli attacchi DoS. Anche l'uso dei file di core dump dovrebbe essere vietato. Nel file */etc/security/limits.conf* modificare le seguenti righe:

```
*    hard  core  0
*    soft  nproc 200
*    hard  nproc 250
```

8. Aumentare la quantità di informazioni registrate nei file di log dal sistema, in special modo far si che i messaggi del kernel vengano registrati e si abbia ampia informazione sui tentativi di accesso. Se l'installazione non ha già previsto di default dei log analoghi, e se si usa il sistema *syslogd + logrotate*, seguire i seguenti passi:
    - Creare i tre file
      /var/log/syslog
      /var/log/kernel
      /var/log/loginlog
    - Aggiungere al file */etc/syslog.conf* le istruzioni che permettono di registrare sui nuovi file rispettivamente gli errori e i warning, i messaggi del kernel e tutti i messaggi riguardanti i login. Inoltre inviare sulle console Alt+F7 e Alt+F8 i dati addizionali:

```
*.warn;*.err                    /var/log/syslog
kern.*                          /var/log/kernel
auth.*;user.*;daemon.none       /var/log/loginlog
*.info;mail.none;authpriv.none  /dev/tty7
authpriv.*                      /dev/tty7
*.warn;*.err                    /dev/tty7
kern.*                          /dev/tty7
mail.*                          /dev/tty8
*.*                             /dev/tty12
```

    - Aggiungere le righe seguenti al file */etc/logrotate.d/syslog*

```
/var/log/kernel {
     postrotate
          /usr/bin/killall –HUP syslogd
```

```
    endscript
}
```

più le analoghe per */var/log/syslog* e */var/log/loginlog*

- Far ripartire il servizio di syslog con:
  # `/etc/init.d/syslogd restart`
  Logrotate viene lanciato periodicamente da crond nei comandi eseguiti in */etc/cron.weekly* o */etc/cron.daily*

9. Disabilitare il compilatore gcc, permettendo l'accesso solo a root
   # `chmod 700 /usr/bin/gcc`

10. Limitare il numero di utenti che possono diventare root con il comando su. Nel web server nessun utente dovrebbe essere autorizzato a diventare root, tranne l'amministratore di sistema:
    - Modificare il file */etc/pam.d/su* e eliminare il commento dalla linea seguente:
      `auth required /lib/security/pam_wheel.so use_uid`
    - Aggiungere al gruppo wheel gli account autorizzati, ossia solo l'account dell'amministatore:
      `# usermod -G10 admin`
    - Infine modificare il file */etc/securetty* commentando con # tutte le righe tranne quella relativa a *tty1* in modo che l'unica console disponibile per il root login sia la prima.

11. Tipicamente ogni installazione crea un certo numero di account e gruppi inutili, che aumentano la possibilità di intrusione nella macchina. E' possibile, anzi necessario eliminare tutti gli account e i gruppi che si riferiscono a servizi non installati. In assenza di altre particolari necessità si può arrivare ad avere una configurazione di partenza con i soli utenti *root, bin, daemon, sync, nobody* e con i gruppi *root, bin, daemon, sys, tty, disj, cdrom, floppy, mem, kmem, wheel, man, nobody, users, floppy, slocate, utmp*.

12. E' necessario anche analizzare i servizi che sono stati attivati dalla installazione e rimuovere quelli non necessari con il comando
    #`chkconfig –delete <nome servizio>`

13. Sarebbe opportuno, se questo non crea troppi disagi all'utenza, non permettere il login interattivo al web server: per la costruzione delle pagine personali basta trasferire i file necessari in una directory, accessibile seguendo i passi qui descritti:

- Rendere non scrivibile la directory degli utenti e i files di configurazione dell'ambiente e predisporre la directory per l'upload dei file.
- Disabilitare la possibilità per l'utente di costruirsi un ambiente di shell proprio, impostando nel file */etc/ssh/sshd_config* la variabile
  `PermitUserEnvironment=no`
- In */etc/passwd* al posto della shell indicare per gli utenti il path completo al server sftp. In questo modo il login sarà possibile ma qualsiasi comando interattivo causerà la chiusura della connessione. Una alternativa più elegante consiste nell'usare *rssh* [13], una shell che permette l'esecuzione del solo sftp server. Una volta installata nel sistema bisogna indicarla come shell degli account utenti e modificare il file */usr/local/etc/rssh.conf* per abilitare sftp.

APPENDICE C – FILE DI CONFIGURAZIONE

/etc/conf.d/uml.start (host)

```
#!/bin/bash

tunctl -u uid -b

echo 1 > /proc/sys/net/ipv4/ip_forward

ifconfig tap0 ip.device.tap netmask 255.255.255.0

route add -host 192.168.0.1 dev tap0

echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp

arp -Ds 192.168.0.1 eth0 pub
```

/etc/conf.d/eth0l.start (UML)

```
#!/bin/bash

ifconfig eth0 192.168.0.1 up

route add default gw 192.168.0.254
```

/etc/conf.d/firewall.sh

```
#!/bin/bash

IPTABLES=/sbin/iptables

#creazione delle catene utente

$IPTABLES -N ADDRESS-FILTER;
$IPTABLES -N REJECT-PKT;
$IPTABLES -N SYN-FLOOD;

# Permette il traffico di loopback

$IPTABLES -A INPUT -i lo -j ACCEPT

# blocca i tentativi di spoof del loopback

$IPTABLES -A INPUT -s 127.0.0.0/8 -j LOG --log-prefix "SPOOFED-LOOPBACK: "
$IPTABLES -A INPUT -s 127.0.0.0/8 -j DROP
$IPTABLES -A INPUT -d 127.0.0.0/8 -j LOG --log-prefix "SPOOFED-LOOPBACK: "
$IPTABLES -A INPUT -d 127.0.0.0/8 -j DROP

# blocca i tentativi di spoof dell'indirizzo host

$IPTABLES -A INPUT -s ip.pubblico.server.host -j LOG --log-prefix "SPOOFED-IP: "
$IPTABLES -A INPUT -s ip.pubblico.server.host -j DROP

# Blocca gli attacchi SYN-FLOOD

$IPTABLES -A INPUT -p tcp -m tcp --syn -j SYN-FLOOD

# Si assicura che la connessione tcp inizi con un pacchetto SYN

$IPTABLES -A INPUT -p tcp -m tcp ! --syn -m state --state NEW -j LOG --log-prefix "SYN-
EXPECTED: "
$IPTABLES -A INPUT -p tcp -m tcp ! --syn -m state --state NEW -j DROP

# permette la continuazione di una sessione già iniziata

$IPTABLES -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

```
# filtra gli indirizzi autorizzati

$IPTABLES -A INPUT -j ADDRESS-FILTER

# permette il ping solo dagli indirizzi filtrati

$IPTABLES -A INPUT -p icmp -m icmp --icmp-type ping -j ACCEPT

# blocca ogni altro tipo di traffico

$IPTABLES -A INPUT -j REJECT-PKT

##########################################################################
#Forward delle richieste Web e ssh
$IPTABLES -P FORWARD DROP
$IPTABLES -A FORWARD -p tcp -i eth0 -d ip.server.uml --dport 80 -j ACCEPT
$IPTABLES -A FORWARD -p tcp -i eth0 -d ip.server.uml --dport 22 -j ACCEPT
$IPTABLES -A FORWARD -p tcp -m state --state ESTABLISHED,RELATED -i tap0 -j ACCEPT

##########################################################################
#Masquerading e NAT
$IPTABLES -t nat -A POSTROUTING -o eth0 -j SNAT --to ip.pubblico.server.host
$IPTABLES -t nat -A PREROUTING -p tcp -i eth0 -d ip.pubblico.server.host --dport 80 -j DNAT --
to ip.server.uml:80
$IPTABLES -t nat -A PREROUTING -p tcp -i eth0 -d ip.pubblico.server.host --dport 22 -j DNAT --
to ip.server.uml:22


##########################################################################
# Definizione della catena di protezione dal syn-flood

$IPTABLES -A SYN-FLOOD -m limit --limit 1/s --limit-burst 4 -j RETURN
$IPTABLES -A SYN-FLOOD -j LOG --log-prefix "SYN-FLOOD: "
$IPTABLES -A SYN-FLOOD -j DROP

##########################################################################
# Definizione della catena di eliminazione dei pacchetti

$IPTABLES -A REJECT-PKT -p udp -m udp --sport 137:138 --dport 137:138 -j DROP
$IPTABLES -A REJECT-PKT -p tcp -m tcp -j LOG
$IPTABLES -A REJECT-PKT -p tcp -m tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A REJECT-PKT -p udp -m udp -j LOG
$IPTABLES -A REJECT-PKT -p udp -m udp -j REJECT --reject-with icmp-port-unreachable
$IPTABLES -A REJECT-PKT -p icmp -m icmp --icmp-type ping -j LOG
$IPTABLES -A REJECT-PKT -p icmp -m icmp --icmp-type ping -j REJECT --reject-with icmp-host-
unreachable

##########################################################################
# Catena di filtraggio degli host autorizzati al ping

$IPTABLES -A ADDRESS-FILTER -s ip.host.autorizzato -j RETURN
$IPTABLES -A ADDRESS-FILTER -j REJECT-PKT
```