

REMUS

REference Monitor for Unix Systems

Giacomo Magnini

`g.magnini@libero.it`

Ivano Alonzi

`alonzi@caspur.it`

Il problema

- Molti processi possono essere forzati ad eseguire istruzioni non *previste*.
- Una tecnica usata comunemente è il *buffer overflow*.
- Il buffer overflow è reso possibile dalla mancanza, in molti linguaggi di programmazione (*ad es.* C e C++), del bound checking.

Le soluzioni

- Diverse soluzioni proposte:
 - StackGuard
 - StackShield
 - Linker modificato (BSDI, OpenBSD)
 - Stack non eseguibile
 - Bound checking degli array
 - Forte tipizzazione (Data Type Enforcement)

Nuovi problemi

- Occorre ricompilare/modificare le applicazioni
- Occorre acquistare nuovi prodotti
- Si penalizzano le prestazioni
- Incompatibilità di alcune applicazioni

Un approccio diverso

- Ogni processo può provocare falle di sicurezza se e solo se *inganna* l'entità che controlla l'accesso alle risorse, ovvero il sistema operativo.
 - Il kernel può non avere sufficienti informazioni per distinguere le richieste lecite da quelle fraudolente.
- Il nostro approccio mira a potenziare il kernel per renderlo più *degnò di fiducia*.
- Le chiamate di sistema sono l'interfaccia ufficiale ai servizi del sistema operativo.
 - **Proponiamo di aggiungere un meccanismo di controllo per l'esecuzione delle chiamate di sistema “critiche”**

Requisiti fondamentali

- Nessuna modifica alle esistenti strutture di dati del kernel
- Nessuna modifica né ricompilazione dei programmi, librerie ed altri componenti software esistenti
- L'impatto sulle prestazioni deve essere limitato (trascurabile per tutti gli scopi pratici)
- L'amministratore di sistema dovrebbe compiere le azioni richieste *una volta per tutte*

I processi privilegiati

I processi in esecuzione con privilegi speciali sono classificati come:

interattivi: Questo è un processo generico lanciato dall'utente *root*.

Sia lo User IDentifier (UID) sia l'Effective User IDentifier (EUID) sono uguali a 0. È necessario evitare che un processo possa entrare a far parte di questa categoria.

background: Questo può essere o un demone partito all'accensione o un processo lanciato periodicamente dal demone *cron* al posto di *root*. Tali processi non hanno un terminale di controllo. Per distinguerli nel kernel:

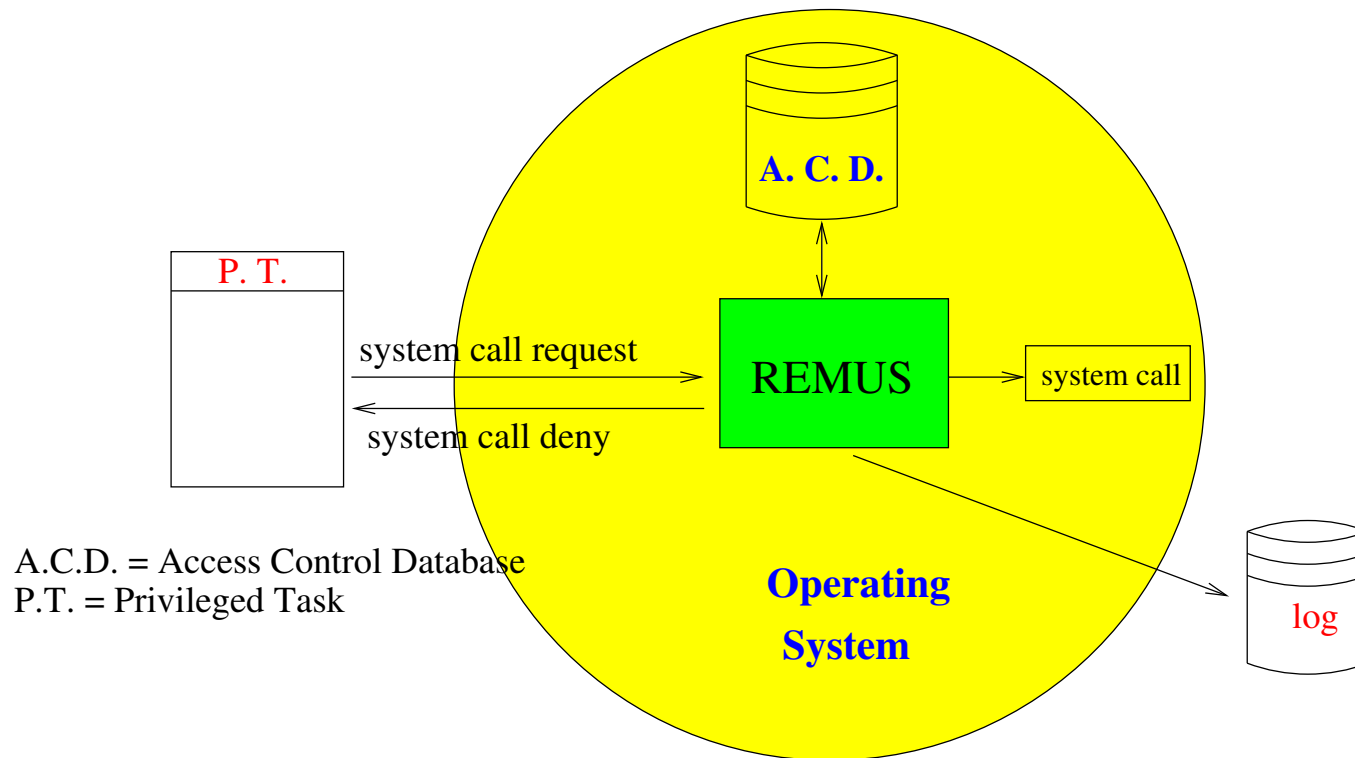
```
#define IS_A_ROOT_DAEMON(proc) !((proc)->euid)&&((proc)->tty==NULL)
```

Ogni tentativo fatto da questi processi per riottenere un terminale di controllo deve essere bloccato.

setuid: Questo è un processo che utilizza il meccanismo standard UNIX per assicurare ai normali utenti degli speciali privilegi su base temporanea. Un processo può essere identificato come setuid a root (EUID=0) tramite il seguente semplice controllo:

```
#define IS_SETUID_TO_ROOT(proc) !((proc)->euid)&&(proc)->uid)
```

Il meccanismo di controllo degli accessi



Analisi

Table 1: Classificazione delle chiamate di sistema Linux

<i>gruppo</i>	<i>funzionalità</i>	<i>gruppo</i>	<i>funzionalità</i>
I	File system, device	VI	Comunicazione
II	Gestione processi	VII	Informazioni di sistema
III	Gestione moduli	VIII	Riservate
IV	Gestione memoria	IX	Non implementate
V	Tempo e contatori		

Analisi (II)

Table 2: Classificazione livello di minaccia

<i>livello minaccia</i>	<i>descrizione</i>
1	Permette di ottenere il pieno controllo del sistema
2	Usata per un attacco denial of service
3	Usata per sovvertire il processo invocante
4	Non è pericolosa

Classificazione chiamate di sistema

Table 3: Livello di minaccia - Funzionalità

<i>livello</i>	<i>gruppo</i>	<i>chiamate di sistema</i>
1	I	open, link, unlink, chmod, lchown, rename, fchown, chown, mount, symlink, fchmod
	II	execve, ptrace, setgid, setreuid, setregid, setgroups, setfsuid, setfsgid, setresuid, setresgid, setuid
	III	init_module

Chiamate di sistema - livello di minaccia 1

chiamate di sistema	parametro pericoloso
chmod, fchmod	un file o una directory di sistema
chown, fchown, lchown	un file o una directory di sistema
execve	un file eseguibile
mount	su una directory di sistema
rename, open	un file di sistema
link, symlink, unlink	un file di sistema
setuid, setresuid, setfsuid, setreuid	UID impostato a zero
setgroups, setgid, setfsgid, setresgid setregid	GID impostato a zero
init_module	moduli maliziosi

Ancora sul livello di minaccia 1

Alcune chiamate di sistema sono considerate al livello di minaccia 1 solo in combinazione con altre (*catene* con livello di minaccia 1):

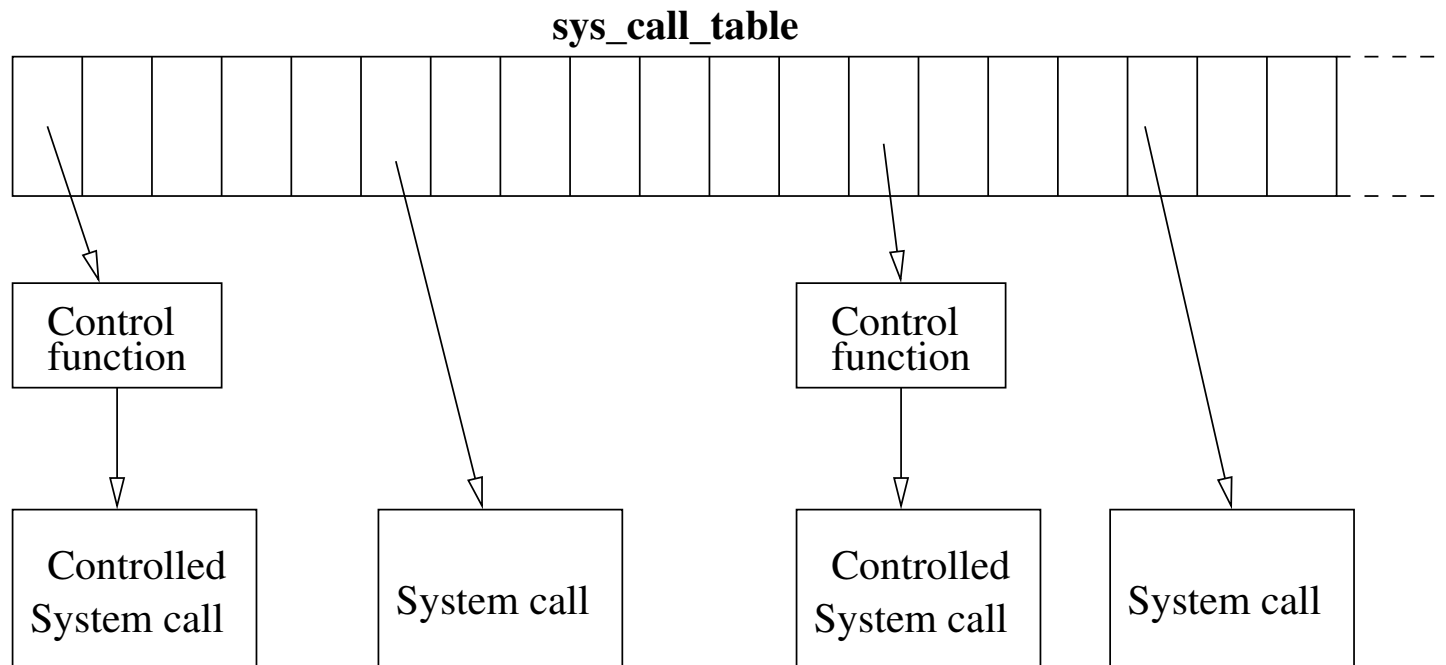
```
unlink('/etc/passwd')
```

```
link('/tmp/mypasswd', '/etc/passwd')
```

produce lo stesso (sgradevole) effetto di

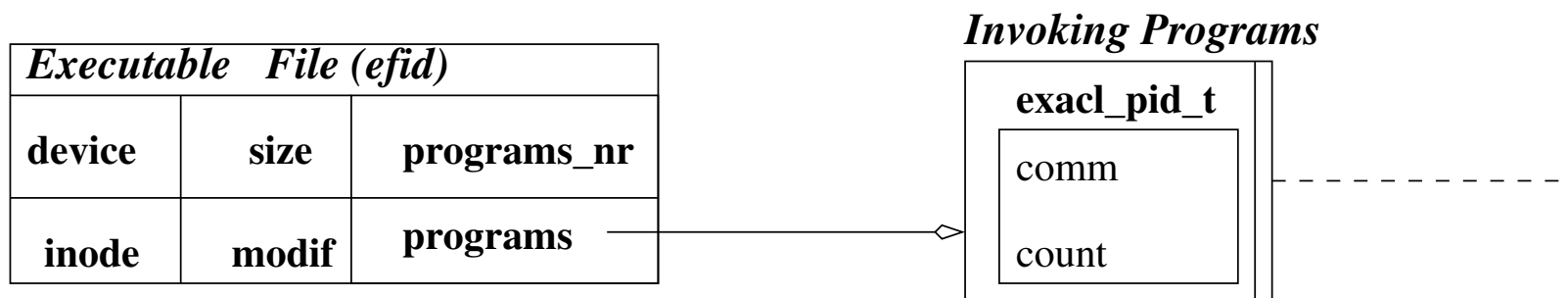
```
rename('/tmp/mypasswd', '/etc/passwd')
```

L'intercettazione delle chiamate di sistema



Un esempio: `execve_acd`

I programmi che possono essere eseguiti (tramite *exec*) da un processo privilegiato hanno un elemento nel DCA. L'elemento contiene le informazioni necessarie ad identificare il programma ed i processi invocanti.



Come trattare i moduli?

- *Problema*

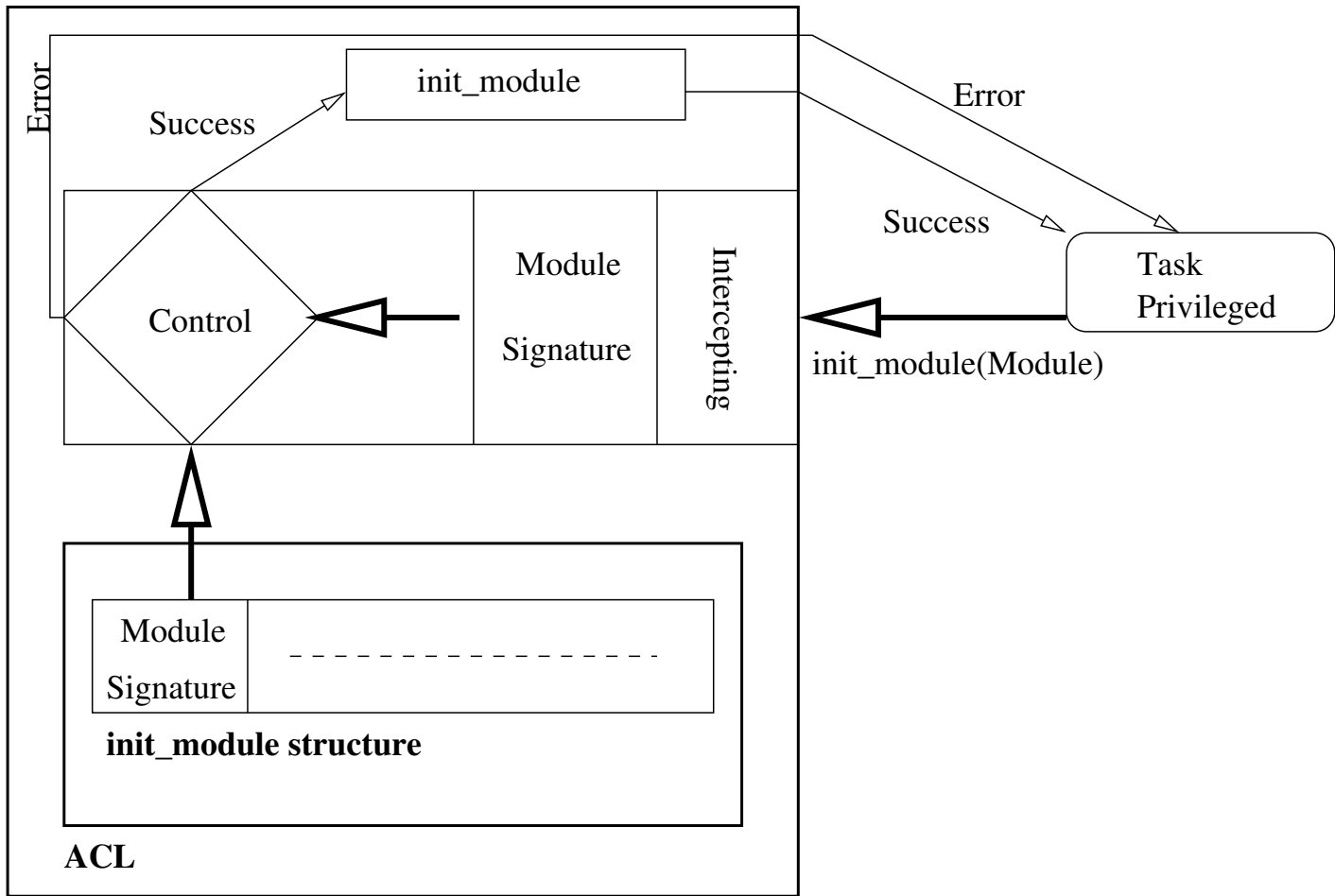
Assicurare che solo i moduli autorizzati dall'amministratore di sistema vengano caricati nel kernel.

- *Soluzione*

Meccanismo di autenticazione basato su immagini dei moduli firmate digitalmente (algoritmo MD5).



Al caricamento del modulo



Kernel + Remus

Gestione del DCA: l'interfaccia /proc

Il contenuto del DCA può essere visualizzato tramite il filesystem `/proc`:

- Ogni chiamata di sistema ha un *nodo* ed una *intestazione* che specifica la configurazione corrente del meccanismo di controllo (quali chiamate di sistema sono controllate e/o bloccate, il livello di debug...)

<code>acl</code>	<code>exec</code>	<code>link</code>	<code>setfsuid</code>	<code>setresuid</code>
<code>chmod</code>	<code>fchmod</code>	<code>mount</code>	<code>setgid</code>	<code>setreuid</code>
<code>chown</code>	<code>fchown</code>	<code>open</code>	<code>setgroups</code>	<code>setuid</code>
<code>create_module</code>	<code>init_module</code>	<code>rename</code>	<code>setregid</code>	<code>symlink</code>
<code>delete_module</code>	<code>lchown</code>	<code>setfsgid</code>	<code>setresgid</code>	<code>unlink</code>

Gestione del DCA: l'interfaccia sysctl

- Per gestire il DCA, usiamo l'interfaccia standard sysctl:
`sysctl -w remus.exec='ADD /usr/bin/procmail sendmail'`
- Ciò permette di avere un front-end comune per tutte le operazioni di gestione del DCA:
 - ADD per aggiungere un elemento
 - DEL per cancellare un elemento
 - CTL per modificare i bit di controllo
- Può essere invocato solo da processi root interattivi con **EUID=0** e **UID=0**
 - **Questa costrizione impedisce ad un demone sovvertito o un programma setuid a root di modificare il DCA**

Prestazioni

- Solo i processi privilegiati devono essere controllati, nessun rallentamento per i processi non privilegiati.
- Nessun rallentamento per i processi privilegiati che girano nello spazio utente.
- Il controllo è limitato alle chiamate di sistema critiche invocate da processi privilegiati.

Conclusioni

- Abbiamo analizzato l'interazione tra le applicazioni privilegiate ed il kernel.
- Un insieme di chiamate di sistema “critiche” è stato isolato e l'esecuzione di queste chiamate viene negato ai processi privilegiati a meno che non ci sia una perfetta corrispondenza del pattern di esecuzione (programma invocante, argomenti...) con il relativo elemento nel DCA.
- Il prossimo passo è l'aggiunta della “capacità di reazione”, inclusa la migrazione del processo sospetto in una *gabbia*.
- Il prototipo corrente è già disponibile su:
<http://remus.sourceforge.net/>

Fine