

Linux Security Modules (LSM)

A cura di Scola Mariano

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

I moduli del kernel

- Tutti i moderni sistemi operativi supportano i cosiddetti moduli del kernel caricabili dinamicamente (LKM)
- Si tratta di porzione di software che può essere caricato a run time al fine di gestire un dispositivo o un device driver

Linux e i moduli

- Da sempre Linux ha supportato i moduli caricabili dinamicamente
- In questo modo si evitava di dover ricompilare il kernel ogni qualvolta si aggiungeva un dispositivo o un device driver

Vantaggi e svantaggi

- Da un lato venivano elogiati i vantaggi forniti dall'utilizzo di porzioni di codice caricate a run-time
- Il problema però consisteva nel fatto che Linux non aveva una infrastruttura che permettesse di gestire il modo in cui i moduli dovevano accedere alle strutture interne del Kernel

La nascita di LSM

- Quello che quindi mancava al sistema operativo Linux, era proprio un'interfaccia tra LKM e le strutture interne del kernel
- La soluzione si presentò nel 2001 con la nascita del progetto Linux Security Modules (LSM), ideato come un leggero framework per la gestione del controllo degli accessi

Evoluzione di LSM

- Inizialmente LSM si è presentato sotto forma di patch per la versione 2.4 del kernel
- Lo sviluppo del progetto è andato avanti e quella che originariamente era una patch, è diventata parte integrante della versione stabile 2.6 del kernel di Linux

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

I consigli di Linus Torvalds

- Secondo Linus Torvalds, il framework LSM nascente doveva essere:
 - **Generico:** LSM deve soltanto fornire l'interfaccia alle strutture interne del kernel. Questo requisito consente di cambiare il modello di sicurezza adottato, semplicemente caricando un nuovo modulo
 - **Semplice:** la complessità del framework deve essere tale da non far degradare le prestazioni del sistema operativo, risultando al tempo stesso il meno invasivo possibile rispetto al kernel

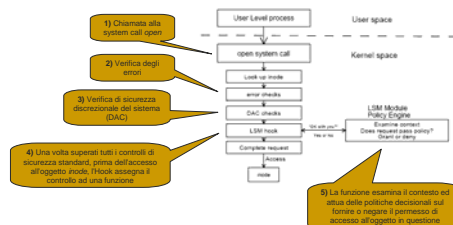
I security hook

- LSM è stato implementato come layer che si pone tra gli oggetti interni del kernel (*task*, *inode*, ecc.) e gli strati superiori che tentano di accedervi
- L'accesso alle risorse viene mediato dal framework LSM tramite degli hook, che sono dei puntatori a funzioni inseriti all'interno delle system call

Funzionamento degli hook

- Quando viene richiesto l'accesso ad una risorsa, gli hook attuano una politica decisionale per l'accesso a tale risorsa
- Infatti, subito prima di accedere alle strutture interne del kernel, l'hook passa il controllo ad un modulo che attua delle politiche di sicurezza, consentendo o negando l'accesso

Esempio



Natura restrittiva degli hook

- La natura degli hook è in genere restrittiva poiché vengono eseguiti solo se il kernel consente l'accesso ad una risorsa
- In genere un hook può soltanto ridurre i privilegi di accesso alle risorse
- Fa eccezione l'hook *capable()* che è autoritativo, nel senso che può 'scavalcare' le politiche di controllo degli accessi discrezionali

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

L'interfaccia LSM

- Fondamentalmente, l'interfaccia LSM è implementata come una tavola di funzioni, detta *security_ops*
- Durante il boot del sistema, questa tavola è inizializzata con le funzioni di default, che implementano le politiche di sicurezza tradizionali di superuser (DAC)
- E' poi compito del particolare modulo andare a sostituire una certa funzione con la propria implementazione, in accordo al tipo di politica di sicurezza che il modulo vuole attuare

Caricamento dei moduli

- Quando viene caricato, un modulo di sicurezza deve registrare le proprie funzioni con il framework LSM, effettuando una chiamata alla funzione *register_security()*
- In questo modo il modulo sostituisce le funzioni di default della tavola *security_ops* con le proprie implementazioni

Scaricamento dei moduli

- Similmente, quando viene scaricato, il modulo effettua una chiamata alla funzione *unregister_security()*
- In questo modo si ripristina la tavola *security_ops* con le funzioni di default

Invasività di LSM

- Una richiesta fondamentale consisteva nel fatto che LSM dovesse essere il meno invasivo possibile
- l'installazione della patch del framework LSM va a modificare il kernel del sistema in due modi principali

Modifiche al kernel

- Aggiungendo dei "Security Field" ad alcune strutture interne del kernel
- Aggiungendo "Security Hook" in vari punti del codice del kernel

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

I security field

- Il framework LSM ha introdotto dei security field in alcune strutture critiche del kernel
- Un security field è un puntatore di tipo *void** che consente ad un modulo LSM di associare delle informazioni aggiuntive, ritenute utili da parte di chi sviluppa i moduli, a strutture interne del kernel

LSM e strutture del kernel

STRUCTURE	OBJECT
<i>task_struct</i>	Task (Process)
<i>linux_binfmt</i>	Program
<i>super_block</i>	Filesystem
<i>inode</i>	Pipe, File, or Socket
<i>file</i>	Open File
<i>sk_buff</i>	Network Buffer (Packet)
<i>net_device</i>	Network Device
<i>kern_ipc_perm</i>	Semaphore, Shared Memory Segment, or Message Queue
<i>msg_queue</i>	Individual Message

- Panoramica delle principali strutture dati del kernel modificate dalla patch LSM, con i corrispondenti oggetti astratti

Esempio

- La struttura *task_struct* è quella in cui sono contenute le informazioni relative ad un processo e al suo stato:

```
linux/sched.h:  
struct task_struct {  
    ...  
    void *security;  
    ...  
};
```

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

I security hooks

- I security hooks sono dei puntatori a funzione che servono sia a gestire i security field sia a mediare l'accesso alle strutture interne del kernel
- Tutti gli hook messi a disposizione dal framework LSM sono indicizzati dalla struttura *security_ops*

La struttura security_ops

```
linux/security.h:  
struct security_operations {  
    ...  
    int (*ptrace) (struct task_struct* parent, struct task_struct* child);  
    ...  
    int (*inode_setattr) (struct dentry *dentry, struct iattr *attr);  
    ...  
};
```

- In questa struttura sono definiti i prototipi di tutti gli hook gestiti dal framework LSM

Funzioni dummy

- Il framework LSM associa di default ad ognuno di questi prototipi una funzione dummy che consente sempre l'accesso alla risorsa richiesta e che restituisce il valore 0:

```
security/dummy.c:  
...  
static int dummy_ptrace (struct task_struct *parent, struct  
    task_struct *child)  
{  
    return 0;  
}  
...
```

Uso dei security hooks

- Un modulo LSM, per utilizzare uno dei security hook definiti, non deve fare altro che registrare le proprie funzioni per gli hook che ha interesse a gestire:

```
my_inode_setattr(struct dentry *dentry, struct iattr *iattr)  
{  
    ...  
}  
...  
static struct security_operations my_security_ops = {  
    ...  
    inode_setattr = my_inode_setattr,  
    ...  
};
```

Categorie dei security hooks

- **Task hooks:** Forniscono un'interfaccia di controllo per le funzioni legate alla gestione dei processi e al controllo delle operazioni
- **Program Loading hooks:** Forniscono un'interfaccia di controllo sugli accessi che consente di effettuare delle verifiche di sicurezza sui programmi prima ancora che siano caricati

Categorie dei security hooks

- **IPC hooks:** Permettono di gestire le politiche di controllo sugli accessi per le funzioni di intercomunicazione tra processi
- **Filesystem hooks:** Consentono di gestire in maniera più granulare i permessi sugli oggetti del filesystem
- **Network hooks:** Forniscono un'interfaccia di gestione sia per le interfacce che per funzioni e oggetti legati al contesto di rete

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

Overhead

- Si intende l'ammontare di degrado delle performance dovuto a qualche fattore esterno al sistema
- Nel nostro caso, siamo interessati a valutare l'overhead dovuto all'utilizzo del framework LSM

Overhead di LSM

- In base ai test effettuati emerge un dato fondamentale: Il framework LSM impone un overhead minimo se comparato con il kernel standard di Linux
- Va sottolineato il fatto che questo overhead è composto sia dall'overhead del framework LSM che dall'overhead dell'attuale politica che bisogna comunque far rispettare

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

Microbenchmark

- Un microbenchmark consiste in una serie di test che misurano le performance di operazioni primitive supportate da un certo sistema
- Non dà però una visione generale delle performance a livello applicazione

Il microbenchmark LMBench

- Si tratta di un software che è stato sviluppato specificamente per misurare le performance di particolari tipi di operazioni, come ad esempio accesso ai file e accesso alla memoria
- LMBench è stato particolarmente efficace nello stabilire e mantenere eccellenti performance riguardo a questi tipi di operazioni

Configurazione di prova

- Nei test eseguiti dal team di sviluppo del progetto LSM, sono stati paragonati un kernel Linux standard 2.5.15 con un kernel Linux standard 2.5.15 con la patch LSM applicata
- Il tutto è stato eseguito su un computer equipaggiato con 4 pentium Xeon 700 mhz, 1 Gb di ram e un disco ultra-wide SCSI

Risultati: Macchina a 4 processori

Test Type	2.5.15	2.5.15-lsm	% Overhead with LSM
total CPU	0.80	0.80	-0.0%
total I/O	0.00	0.00	-0.0%
mem	2.39	2.40	0.1%
system	0.04	0.04	0.0%
select TCP	20	20	0.0%
net	1.18	1.19	0.1%
db bench	2.25	2.26	0.1%
fs bench	1.87	1.87	0.0%
fs test	7.01	7.06	0.1%
fs proc	3.00	3.01	0.1%

Test Type	2.5.15	2.5.15-lsm	% Overhead with LSM
DB CPU	0.00	0.00	-0.0%
DB file system	0.147	0.151	0.1%
DB file system	1.57	1.57	0.0%
DB file system	2.21	2.21	0.0%
memory system	0.00	0.00	-0.0%
fs test bench	0.074	0.080	0.8%
fs test bench	0	0	0.0%

Test Type	2.5.15	2.5.15-lsm	% Overhead with LSM
AF CPU	1.07	1.27	18.4%
AF CPU	89	110	23.4%
AF I/O	217	217	0.0%
fs test bench	0.00	0.00	0.0%
fs test bench	0.00	0.00	0.0%
fs test bench	1.08	1.11	2.8%
fs test bench	1.08	1.11	2.8%
fs test bench	1.08	1.11	2.8%
fs test bench	1.08	1.11	2.8%

- Test LMBench, macchina a 4 processori

Risultati: Macchina a 1 processore

Test Type	2.5.15	2.5.15-lsm	% Overhead with LSM
total CPU	0.80	0.80	0.0%
total I/O	0.00	0.00	0.0%
mem	2.39	2.40	0.1%
system	0.04	0.04	0.0%
select TCP	20	20	0.0%
net	1.18	1.17	-0.1%
db bench	2.25	2.24	-0.1%
fs bench	1.87	1.87	0.0%
fs test	7.01	7.02	0.0%
fs proc	3.00	3.00	0.0%

Test Type	2.5.15	2.5.15-lsm	% Overhead with LSM
DB CPU	0.00	0.00	-0.0%
DB file system	0.15	0.15	0.0%
DB file system	1.57	1.58	0.1%
DB file system	2.21	2.21	0.0%
memory system	0.00	0.00	-0.0%
fs test bench	0.074	0.080	0.8%
fs test bench	0	0	0.0%

Test Type	2.5.15	2.5.15-lsm	% Overhead with LSM
AF CPU	1.07	1.07	0.0%
AF CPU	89	89	0.0%
AF I/O	217	217	0.0%
fs test bench	0.00	0.00	0.0%
fs test bench	0.00	0.00	0.0%
fs test bench	1.08	1.08	0.0%
fs test bench	1.08	1.08	0.0%
fs test bench	1.08	1.08	0.0%
fs test bench	1.08	1.08	0.0%

- Test LMBench: Macchina a 1 processore

Analisi dei risultati

- Dall'analisi dei risultati, emerge il fatto che l'impatto sulle performance è abbastanza alterante
- In alcuni casi le performance del kernel LSM eccedono di poco il kernel standard
- Invece, il 18,4% dell'aumento di performance per "AF Unix" nella tavola 2 è abbastanza strano, e infatti non si è riusciti ad identificare il problema in tale test

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

Macrobenchmark

- Un macrobenchmark consiste di uno o più programmi a scala medio-grande che sono normalmente derivati da applicazioni reali
- Il primo macrobenchmark è quello maggiormente utilizzato nella misurazione del tempo necessario per compilare il kernel di Linux

Configurazione di prova

- Questo test è stato eseguito su una macchina SMP a quattro processori Xeon 700mhz, 1Gb di ram, disco ultra wide SCSI
- Sono stati usati sia un kernel SMP che un kernel UP come fatto anche con i microbenchmark

Risultati

Machine Type	2.5.15	2.5.15-lsm	% Overhead with LSM
4 CPUs	92	92	0%
1 CPU	341	342	0.3%

- Macrobenchmark per la compilazione del kernel di Linux. Tempo in secondi

Analisi dei risultati

- Dai risultati emerge che la macchina a singolo processore esegue il comando `time make -j2 bzImage` con solo lo 0,3% di overhead
- La macchina a 4 processori invece esegue lo stesso comando praticamente senza overhead.
- Questo è sicuramente un risultato eccellente, presentando nel caso peggiore soltanto un decadimento delle performance dello 0,3%

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

Webstone

- Si tratta di un macrobenchmark sviluppato dalla Silicon Graphics per misurare le performance di web server e prodotti hardware
- Grazie al suo utilizzo è stato possibile misurare l'overhead imposto da LSM su un webserver, che è una tipica applicazione server

Sistemi a confronto

- Nei test, sono stati analizzati i dati che mostrano l'overhead sia di un kernel LSM di base sia di un kernel LSM con il modulo *SELinux* caricato, entrambi paragonati al kernel standard
- Tutti i kernel sono stati compilati con il supporto a *Netfilter*, in modo tale da effettuare i test partendo dalle stesse condizioni

Configurazione di prova

- I test sono stati eseguiti su una macchina a doppio processore Celeron 550Mhz con 384Mb di ram.
- La scheda di rete è una Gigabit Netgear GA302T su un bus PCI a 32bit e a 33Mhz.
- Il webserver utilizza Apache 1.3.22-0.6, aggiornato a Redhat 6.2

Risultati: LSM

Connection rate measured in connections per second.

Number of clients	Server connection rate 2.5.7	Server connection rate 2.5.7+SEL	% Overhead
8	918.58	879.08	4.97%
16	917.64	869.79	5.21%
24	917.44	872.28	4.92%
32	918.91	876.27	4.65%

Connection rate measured in connections per second.

Number of clients	Server connection rate 2.5.7	Server connection rate 2.5.7+SEL	% Overhead
8	1208.53	1157.29	7.53%
16	1208.74	1117.41	7.39%
24	1214.24	1159.13	4.95%
32	1207.30	1123.89	6.74%

- Risultati UP Webstone di LSM paragonato al kernel standard
- Risultati SMP Webstone di LSM paragonato al kernel standard

Risultati: SELinux

Connection rate measured in connections per second.

Number of clients	Server connection rate 2.5.7	Server connection rate 2.5.7+SEL	% Overhead
8	918.58	766.53	16.4%
16	917.64	766.44	15.7%
24	917.44	785.56	14.6%
32	918.91	784.80	14.8%

Connection rate measured in connections per second.

Number of clients	Server connection rate 2.5.7	Server connection rate 2.5.7+SEL	% Overhead
8	1208.53	948.56	21.3%
16	1208.74	949.74	21.3%
24	1214.24	952.28	21.4%
32	1207.30	956.76	20.1%

- Risultati UP Webstone di SELinux paragonato al kernel standard
- Risultati SMP Webstone di SELinux paragonato al kernel standard

Analisi dei risultati

- Dall'analisi dei dati, emerge che i risultati sono abbastanza critici quando si utilizzano gli hook per *Netfilter*
- Sicuramente il 5-7% dell'overhead osservato per il benchmark LSM nelle tavole 5 e 6 è maggiore di quello che si vorrebbe
- Infatti, un esperimento separato configurato con LSM e *Netfilter*, ma senza gli hook LSM di *Netfilter* mostra un più preferibile 1-2% di overhead sulle performance

Overview

- Introduzione
- Architettura generale
- Implementazione
 - Security Field
 - Security Hooks
- Prestazioni
 - LMBench
 - Compilazione
 - Webstone
- Conclusioni

La sicurezza base di Linux

- le misure di sicurezza del kernel base di Linux in molti casi risultano non adeguati ad offrire un buon livello di sicurezza
- Sono stati sviluppati molti progetti di sicurezza, ma nessuno è riuscito ad emergere

Il progetto LSM

- Il progetto LSM ha fornito un'interfaccia standard utilizzabile per il caricamento dei moduli per il rafforzamento della sicurezza
- Quest' interfaccia è abbastanza ricca da soddisfare un'ampia varietà di moduli di sicurezza

Pregi di LSM

- L'interfaccia LSM impone una minima invasività al codice sorgente di Linux e un minimo overhead sulle performance del kernel
- LSM è stato implementato come patch per le versioni stabili 2.4 e 2.5 del kernel standard di Linux, ed è oggi parte integrante della versione 2.6

Riferimenti

- "LSM overview"
<http://www.nsa.gov/sellinux/papers/module/x44.html>
- "Linux Security Modules: General Security Support for the Linux Kernel"
Chris Wright, Crispin Cowan, James Morris, Stephen Smalley and Greg Kroah-Hartman
http://www.kroah.com/linux/talks/usenix_security_2002_lsm_paper/lsm.pdf
- "Linux Security Module Framework"
Chris Wright, Crispin Cowan, James Morris, Stephen Smalley and Greg Kroah-Hartman
http://www.kroah.com/linux/talks/ols_2002_lsm_paper/lsm.pdf
-
- "Using the Kernel Security Module Interface"
Greg Kroah-Hartman
<http://www.linuxjournal.com/article.php?sid=6279>