

Introduzione a SELinux

a cura di Giorgio Zanin

Sicurezza dei dati e delle reti

a.a. 2002/2003

Introduzione

- I sistemi devono essere flessibili nel loro supporto alle security policy
- Flessibilità significa:
 - Controllare la propagazione dei permessi (modalità di accesso)
 - Utilizzare permessi “fine-grained”
 - Supportare la revoca di permessi precedentemente assegnati
- Il controllo sulla propagazione dei permessi: ogni decisione di sicurezza implica una precedente consultazione della policy
- Permessi molto “fini” e supporto alla revoca: meccanismi direttamente integrati nei componenti del sistema
- Flask: architettura che permette il supporto flessibile alle security policy

Flask

- 3 componenti principali:
 - Security Server (SS)
 - Object managers (Oms)
 - Access Vector Cache (AVC)
- SS: prende le decisioni di sicurezza
- Oms: impongono le decisioni di sicurezza prese dal SS
- Separazione tra logica di decisione e logica di realizzazione
- AVC: fa il caching delle decisioni prese per aumentare le prestazioni
- È possibile modificare o addirittura sostituire il SS senza alterare le altre componenti del sistema

Etichette di oggetti I

- Tutti i soggetti e gli oggetti sono etichettati con degli attributi di sicurezza: security context
- La struttura dei security context dipende dalla policy usata
- Deve essere mantenuta l'associazione tra security context ed oggetti
- 2 tipi di dati indipendenti dalla politica:
 - Security Context (SC): stringa di dimensione variabile interpretata da un'applicazione o dall'utente che abbiano una conoscenza della security policy; I campi dipendono dalla particolare security policy
 - Security Identifier (SID): valore di dimensione fissa che può essere interpretato solo dal SS ed è mappato su un particolare SC; la sua conoscenza non garantisce alcuna autorizzazione
- Tutte le strutture interne sono conosciute ed interpretate solo dal SS
- I SID permettono agli OMs di agire indipendentemente dal contenuto e dal formato dei SC

Etichette ed oggetti II

- Alla creazione di un oggetto viene assegnato un SID che rappresenta univocamente un SC nel quale viene creato un oggetto
- Il contesto dipende in genere dal client che richiede la creazione e dall'ambiente nel quale l'oggetto viene creato:
 - File: soggetto che richiede la creazione e directory contenitrice
 - Processo: soggetto che richiede la creazione e file eseguibile
- La computazione del SC per un oggetto può dipendere dalla policy: non può dipendere dagli Oms; è prerogativa del SS

Security Server

- Incapsula la logica della security policy
- Effettua le decisioni relative alla sicurezza
- 2 tipi di decisioni di sicurezza:
 - Decisioni di accesso se un permesso è garantito tra due entità
 - Decisioni di etichettatura: attributi di sicurezza da assegnare ai nuovi oggetti
- Mantiene il mapping tra SC e SID
- La sua implementazione e qualsiasi linguaggio di configurazione della policy che può supportare, non sono specificate dall'architettura
- Fornisce un'interfaccia per gli OMs e per applicazioni sicure

Object Managers

- Sono sottosistemi responsabili per la definizione di un meccanismo per assegnare etichette ai loro oggetti
- Sono indipendenti dalla specifica security policy utilizzata
- Possono registrarsi presso il SS per ricevere notifiche di cambiamenti della security policy; ogni OM deve definire routine di gestione dei cambiamenti della policy

Access Vector Cache

- Nell'implementazione più semplice: una richiesta degli OMs al SS per ogni decisione di sicurezza
- AVC permette agli OMs di fare il caching delle decisioni di accesso prese dal SS per minimizzare l'overhead
- È possibile anche fare il caching di più decisioni di quante siano state richieste
- Il SS restituisce le decisioni per insiemi di permessi negli access vector
- Un access vector è un insieme di permessi correlati per la coppia di SIDs forniti al SS

SELinux

- Implementazione di Flask in Linux ad opera dell'NSA americana
- Il SS e gli OMs sono semplicemente sottosistemi del kernel
- Il SS di SELinux definisce una policy che è combinazione di:
 - TE
 - RBAC
 - IBAC
 - MLS (opzionale)
- Il SS ha un linguaggio di configurazione associato
- La configurazione scritta nell'apposito linguaggio viene compilata con un programma in una rappresentazione binaria e letta dal SS al boot
- Security Context: Identità + Ruolo + Tipo + livello MLS (opzionale)
- I ruoli sono rilevanti solamente per i processi

Eredità e transizioni di etichette

- In SELinux il SS può essere configurato per cambiare ruolo e dominio di un processo: ruolo e dominio del processo, tipo del programma
- Per default ruolo e dominio del processo non cambiano all'esecuzione dei programmi
- In SELinux il SS può essere configurato per usare tipi specificati per i nuovi file in base al dominio del processo, al tipo della directory contenitrice e al tipo del file
- Un nuovo file eredita il tipo della directory contenitrice per default
- Pipe, file descriptor e socket ereditano il SID del processo creatore, output message il SID del socket che invia
- Gli oggetti sono divisi in classi

Classi e permessi I

- Classe: indica il tipo dell'oggetto: file regolare, directory, processo, socket TCP...
- Ogni classe di oggetti ha un insieme di permessi associato: access vector
- Un differente permesso per ogni servizio
- Quando un servizio accede più oggetti, si controllano i permessi per ciascun oggetto
- Classi di oggetti: diversi permessi per i vari tipi di oggetti in base ai servizi supportati dagli oggetti

Classi e permessi II

- L'accesso ad un device special file è distinto da quello ad un file regolare (lo stesso per socket raw IP e TCP, ...)
- Per i processi:
fork, **transition**, sigchld, sigkill, sigstop, signull, signal, ptrace, getsched, setsched, getsession, getpgid, setpgid, getcap, setcap, share
- Per i file:
ioctl, read, write, execute, create, getattr, setattr, lock, relabelfrom, relabelto, append, unlink, link, rename, **execute**, swapon, quotaon, mouton, execute_no_trans, **entrypoint**
- Per le directory:
quelli di file, add_name, remove_name, reparent, search, rmdir
- Esistono delle macro per riunire in gruppi insiemi di permessi di una classe che comunemente devono essere garantiti o negati in gruppo
- Esistono anche macro per insiemi di classi

Cambiamenti di policy

- Il SS di SELinux fornisce un'interfaccia per cambiare la configurazione della security policy a runtime
- Al caricamento della nuova configurazione il SS:
 - Aggiorna il mapping dei SID
 - Invalida ogni SID non più autorizzato
 - Resetta la AVC
- I controlli su processi e oggetti con SID non validi falliscono: no accesso da parte dei processi e sugli oggetti
- Il supporto per la rietichettatura ancora non implementato

Modello TE

- Modello tradizionale del TE:
 - classi di equivalenza per soggetti ed oggetti
 - 2 matrici di accesso
 - Utenti autorizzati ad agire all'interno di determinati domini
- Controlli forti sull'esecuzione del programma e sulla migrazione di dominio
- Tipi associati ai programmi; la matrice degli accessi:
 - quali tipi possono essere eseguiti da ciascun programma
 - Quali tipi possono essere eseguiti per entrare inizialmente in un dominio
- Dominio associato ad uno o più programmi entrypoint
- Differenze in SELinux:
 - Un unico attributo tipo per processi e oggetti: una sola matrice
 - Utilizzo di classi di sicurezza: le decisioni dipendono dai tipi e dalle classi
 - Gli utenti non associati ai tipi (domini): modello RBAC

Regole TE I

- Dichiarazione di attributi:

```
attribute nomeAttributo;  
attribute domain;  
attribute privuser;  
attribute privdomain;
```

- Dichiarazione di tipi:

```
type nomeTipo opz_alias opz_attributi;  
  
type sshd_t, domain privuser, privrole, privlog, privowner;  
type sshd_exec_t, file_type, exec_type, sysadmfile;  
type sshd_tmp_t, file_type, sysadmfile, tmpfile
```

Regole TE II

- Regole di vettore degli accessi:

```
allow source_t target_t: class permissions
```

```
allow sshd_t sshd_exec_t: file {read execute entrypoint};
```

```
allow sshd_t sshd_tmp_t: file {create read write getattr  
    setattr link unlink rename };
```

```
allow sshd_t user_t: process transition;
```

- Asserzioni:

```
neverallow source_t target_t: class permissions
```

```
neverallow domain ~domain:process transition
```

```
neverallow local_login_t ~login_exec_t:file entrypoint
```

Transizioni TE I

- È possibile definire delle regole di transizione per i security context
- Una transizione di security context avviene in due casi:
 - per l'attribuzione di etichette ai file creati
 - per l'attribuzione di un'etichetta ad un processo trasformato
- Un processo transisce di dominio solo all'invocazione della system call `execve()` quando ci sono regole apposite
- Perché un processo possa transire di dominio:
 - Deve essere definito l'entrypoint del nuovo dominio
 - Deve essere autorizzato a transire nel nuovo dominio
 - Il suo dominio e quello nuovo devono avere determinati altri permessi
- Le transizioni possono essere automatiche

Transizioni TE II

- Le regole per attribuire permessi ad un tipo le conosciamo
- La regola che permette una transizione di tipo è `type_transition`
`type_transition source_t target_t:class new_t`
`source_t` è il tipo di partenza
`target_t` è il tipo dell'entrypoint del nuovo dominio
`class` è `process`
`new_t` è il nuovo dominio nel quale gira il processo trasformato

```
type_transition initrc_t sshd_exec_t:process sshd_t;  
type_transition sshd_t shell_exec_t:process user_t;
```

- La regola può essere applicata anche per etichettare nuovi file
- Poichè ogni regola `type_transition` prevede precedenti regole `allow`, vengono utilizzate delle apposite macro

Macro per le transizioni di domini

```
domain_trans(parent_domain, program_type, child_domain)
```

```
    allow $1 $3:process transition
```

```
    allow $1 $2: file x_file_perms
```

```
    allow $3 $1: process sigchld
```

```
    allow $3 $1:fd use;
```

```
    allow $1 $3:fd use;
```

```
    allow $3 $1:fifo_file rw_file_perms
```

```
    allow $3 $2:file rx_file_perms
```

```
    allow $3 $2:file entrypoint
```

```
domain_auto_trans(parent_domain, program_type, child_domain)
```

```
    domain_trans($1,$2,$3)
```

```
    type_transition $1 $2:process $3;
```

Modello RBAC

- RBAC standard:
 - Autorizza gli utenti ad agire in determinati ruoli
 - Assegna insiemi di permessi a ruoli
- In SELinux:
 - Utenti assegnati a insiemi di ruoli
 - Ogni ruolo autorizzato ad un insieme di domini TE
- Opzionalmente relazione di dominanza: gerarchia di ruoli
- L'assegnamento di permessi è lasciato principalmente al TE
- L'attributo ruolo è in ogni SC ma ha senso solamente per i processi: object_r per gli oggetti
- Transizioni di ruolo possibili ma limitate: transizioni TE

Regole RBAC

- Dichiarazione di ruolo:

```
role nomeruolo types tipi;
```

```
role user_r types { user_t user_netscape_t }  
role sysadm_r types {sysadm_t run_init_t }
```

- Dominanza di ruolo:

```
dominance { ROLE senior; ROLE junior }
```

- Transizioni di ruolo:

```
allow ruolo_corrente nuovoruolo;
```

```
allow system_r { user_r sysadm_r };  
allow user_r sysadm_r;
```

Modello IBAC

- L'identità di Linux non è sicura:
 - cambia per esprimere un cambiamento dei permessi o privilegi
 - non cambia per indicare cambiamenti di utenti
- Uid Linux cambiate dal superuser e dai processi setuid
- SELinux mantiene un attributo identità nei SC indipendente dall'identità Linux
- Identità SELinux ortogonale a Identità Linux: il cambiamento di una non si riflette sull'altra
- Il cambiamento di Identità SELinux è rigidamente controllato e limitato dalla policy
- Cambiamento limitato a certi programmi come: login, crond, sshd; versioni modificate per le nuove chiamate
- Più utenti Linux in una stessa identità SELinux
- In sostanza: Identità → Ruoli → Tipi

Regole IBAC

- Dichiarazione identità:

```
user nomeutente roles ruoli
```

```
user system_u roles system_r;
```

```
user root roles {user_r sysadm_r};
```

```
user giorgio roles user_r;
```

```
user marco roles user_r;
```

Constraints

- Specificano vincoli sui permessi nella forma di espressioni booleane
- L' espressione booleana puo' riguardare identita', ruoli e tipi di una coppia di security context

```
constrain class permissions espressione
```

```
constrain process transition (u1 == u2 or t1 == privuser)
```

```
constrain process transition (r1 == r2 or r1 == privrole)
```

```
constrain dir_file_class_set { create relabelto relabelfrom }  
    (u1 == u2 or t1 == privowner)
```

Bibliografia

- [LS01a] Loscocco, P. A., and Smalley, S. D. 2001a. Integrating flexible support for security policies into the Linux operating system. *NSA Technical Report*, Feb.
- [LS01b] Loscocco, P. A., and Smalley, S. D. 2001b. Meeting critical security objectives with Security-Enhanced Linux. In *Proceedings of the 2001 Ottawa Linux Symposium*, July
- [LSMTTF98] Loscocco, P. A., Smalley, S. D., Muckelbauer, P. A., Taylor, R. C., Turner, S. J., and Farrel, J. F. 1998. The Inevitability of failure: The flawed assumption of security in modern computing environments. In *Proceedings of the 21st National Information Systems Security Conference*, pp 303-314, Oct.
- [S02] Smalley, S. D. 2002. Configuring the SELinux Policy. *Nai Labs Report #02-007*, June, pg 1-25
- [SF01] Smalley, S. D., and Fraser, T. 2001. A security policy configuration for the Security-Enhanced Linux. *NAI Labs Technical Report*, Feb.
- [SSLHAL99] Spencer, R., Smalley, S. D., Loscocco, P., Hibler, M., Andersen, D., and Lepreau, J. 1999. The Flask Security Architecture: System support for diverse security policies. In *Proceedings of the 8th USENIX Security Symposium*, Aug, pp 123-139

