Tecnologie e sistemi per la sicurezza LS

L'implementazione di modelli di sicurezza evoluti nel sistema operativo Linux

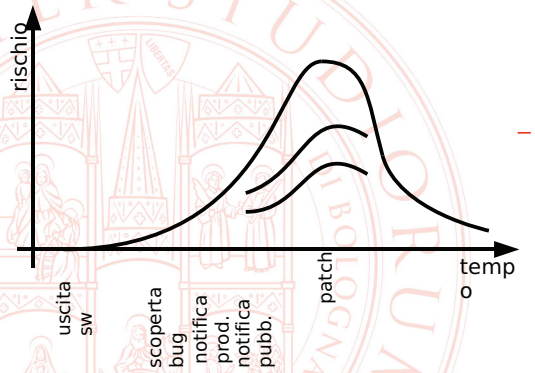
19 novembre 2008
Marco Prandini
DEIS - Università di Bologna

Linux - generalità

- Linux, come tutti i sistemi operativi di più ampia diffusione, soffre di due problemi strutturali che contribuiscono alla difficoltà della sua messa in sicurezza:
  - scarso controllo sull'utilizzo appropriato della memoria e dei meccanismi di comunicazione interprocesso, a vantaggio dell'efficienza
    - complessità del software = maggiore difficoltà di verifica della correttezza, in presenza di un S.O. che non la impone
  - inefficace limitazione dei privilegi di accesso alle risorse, secondo il modello DAC, a vantaggio della semplicità di configurazione
    - assenza di livelli di privilegio intermedi tra l'utente comune e l'utente amministrativo

Hardening

- L'approccio tradizionale alla soluzione delle vulnerabilità è il cosiddetto *penetrate and patch*, che soffre di due gravi problemi:
  - la "pezza" è spesso applicata localmente perdendo di vista il sistema, e può finire con l'introdurre problemi tanto gravi quanto quelli che risolve
  - il ciclo di vita della vulnerabilità lascia comunque aperte aree significative di rischio



Hardening

- Un approccio più efficace deve prevedere:
  - prevenzione
  - rilevazione
  - contenimento
- Tali funzionalità possono essere implementate a vari livelli
  - infrastruttura di rete (firewall, ids, ips)
  - meccanismi di controllo dell'esecuzione dei processi (logging, memory checking, randomization, nx-pages, ...)
  - meccanismi di controllo dell'accesso alle risorse (acl, sandboxing, ...)

Un esempio di exploit

- Un esempio classico: Apache + ptrace
  - una vulnerabilità di *OpenSSL* permetteva di eseguire codice arbitrario da remoto con i privilegi dell'utente apache
  - una vulnerabilità del programma SUID *ptrace* permetteva di acquisire diritti di root eseguendolo da utente standard
- Le vulnerabilità derivano dall'assenza di meccanismi in grado di garantire il corretto comportamento del software, e sono rimaste utilizzabili a lungo prima del rilascio di una patch (prevenzione)
- La possibilità di sfruttare la seconda dalla prima (contenimento) deriva dalla debolezza del modello DAC che porta a concedere privilegi eccessivi prima all'utente apache e poi all'utente root

Modelli per l'integrazione in Linux

- L'approccio iniziale all'hardening di Linux consisteva in patch per modificare il funzionamento di parti specifiche del S.O.
- Un sistema ben progettato deve consentire di registrare "funzioni di sicurezza" attivate ad ogni invocazione di qualsiasi operazione sensibile
  - approccio suggerito da Linus Torvalds dopo la presentazione di SELinux, come prerequisito per integrarlo nel kernel
- LSM (Linux Security Modules)
  - Sviluppato da Immunix ed altri sul kernel 2.5 dal 2001
  - Incluso ufficialmente nel kernel 2.6
  - http://lsm.immunix.org/lsm_doc.html
  - Abbandonato nel 2006

LSM

- Cosa aggiunge:
  - campi di sicurezza alle strutture dati del kernel
  - chiamate a funzioni hook per utilizzare i campi di sicurezza nei punti critici del codice del kernel
  - funzioni per de/registrare moduli di sicurezza
- In che modo:
  - i campi sono semplicemente puntatori a void
  - ogni hook è un puntatore in una tabella *security_ops*,
    - all'inizializzazione referenzia funzioni che implementano la logica utente/superutente standard
  - un modulo può implementare funzionalità di stacking di altri moduli

LSM - critiche

- Benchè il principio sembrasse solido, LSM fu aspramente criticato sia sul fronte “politico” che su quello tecnico da molti dei più autorevoli esperti di hardening, tra cui gli autori di grsecurity e RSBAC (descritti in seguito)
- Alcuni commenti:
  - “the whole hook design is broken, because all kernel data gets exposed to any module that likes to register - what an invitation to root kit authors. LSM is too low level and kernel version dependent, and it provides no support for more than one module. IMHO, the LSM project’s major fault is that it accepted Linus’ order what hooks should look like, regardless of known security principles. -- Amon Ott”
  - “It seems to be the case now that LSM was simply a ploy to get SELinux into the kernel. Anyone wishing to develop their own security system are now being told to implement it within the SELinux framework, not the LSM framework. -- Brad Spengler”

Progetti per l'hardening di Linux

- Sono disponibili diversi progetti per integrare funzionalità di prevenzione, rilevazione e contenimento degli attacchi, alcuni implementati come patch, altri come LSM/SELinux. I più noti sono:
  - Linux Capabilities
  - PaX
  - Openwall
  - Rule Set Based Access Control (RSBAC)
  - Linux Intrusion Detection System (LIDS)
  - grsecurity
  - SELinux

Capabilities

- Sistema di controllo dell'accesso originariamente integrato nel kernel, ora disponibile come LSM
- Implementa le POSIX Capabilities, come definite dalla bozza 1003.1e (proposta nel 1985 e mai trasformata in standard - ritirata nel 1998)
- Obiettivo: limitare i poteri dei processi del superutente
  - associa ad ogni processo tre set di bit (Inheritable, Permitted, Effective)
    - permette un'operazione solo se il corrispondente bit nel set E è 1
    - un "1" nel set P consente di porre ad "1" il bit corrispondente del set E
    - un processo lanciato con exec riceve un set P'=P&I del padre
- Esempi di capabilities (30 al kernel 2.6.14): CAP_CHOWN, CAP_KILL, CAP_DAC_OVERRIDE, CAP_SETUID, CAP_NET_BIND_SERVICE, ...

PaX

- È sviluppato indipendentemente all'interno del progetto grsecurity
- Mira alla rilevazione e prevenzione delle intrusioni agendo sui meccanismi di basso livello più comunemente utilizzati per
  - introdurre ed eseguire codice arbitrario su di un sistema
    - stack overflow, shellcode injection
  - eseguire codice esistente in ordine diverso dal previsto e/o far elaborare dati arbitrari a codice esistente
    - return into libc
- Le principali contromisure fornite da PaX, che rendono impossibili o molto vistosi i tentativi di attacco, sono:
  - Address Space Layout Randomization
  - NOEXEC

grsecurity

- Patch per il kernel di linux che realizza un sistema RBAC
- Un singolo file di configurazione definisce:
  - ruoli e loro modalità di associazione agli utenti
  - domini (ruoli che raggruppano più utenti o gruppi di sistema)
  - soggetti
    - flag di protezione del soggetto
    - capabilities
    - possibilità di transizione
  - oggetti
    - sono elencati per ogni soggetto, insieme ai diritti che il soggetto vi detiene

grsecurity – sintassi delle ACL

```
<path of subject process> <optional subject modes> {
   <file object> <optional object modes>
   [+|-]<capability>
   <resource name> <soft limit> <hard limit>
   connect {
      <ip>/<netmask>:<low port>-<high port> <type> <proto>
   }
   bind {
      <ip>/<netmask>:<low port>-<high port> <type> <proto>
   }
}
```


grsecurity – esempio di ACL

```
/usr/sbin/apache oXA {
   /usr/share read
   /etc/grsec hidden
   /tmp rwx
   /var/log/apache append
   /var/run/apache.pid write
   ...
   -CAP_ALL
   +CAP_DAC_OVERRIDE
   +CAP_KILL
   +CAP_SETGID
   +CAP_SETUID
   +CAP_NET_BIND_SERVICE
   RES_CRASH 1 10m
}
```

Override ACL inheritance for this process;

Enable the RANDEXEC feature of PaX on this subject;

Protect the shared memory of this subject.

```
connect {
   0.0.0.0/0:53 dgram udp
}
bind {
   0.0.0.0/0:80 stream tcp
}
```


Configurazione di grsecurity/Pax

- Gran parte della configurazione di PaX e grsecurity può essere decisa all'atto della compilazione del kernel, selezionando tra le miriadi di controlli possibili quali attivare
- Si può avviare il sistema in modo che i controlli di grsecurity siano personalizzabili attraverso il filesystem /proc, o attivati in modo immutabile al boot
- grsecurity supporta il *learning mode*
  - per ogni soggetto viene creata una ACL che concede solo i diritti necessari ad eseguire le operazioni sugli oggetti osservate durante la fase di learning
  - deve essere svolto in un ambiente controllato e sicuro!
  - è consigliabile comunque un'accurata revisione

Openwall

- Il progetto Openwall è una collezione piuttosto variegata di strumenti, che comprende:
  - un patch set per il kernel di linux (2.4) che mira sia a correggere comportamenti insicuri del kernel che ad implementare nuove funzionalità (stack non eseguibile, limitazione delle funzionalità delle directory temporanee, limitazione delle risorse concesse ai processi)
  - strumenti per il controllo della qualità delle password
  - versioni sicure di alcuni servizi di sistema
- È disponibile una distribuzione rpm-based (Owl) che integra tutte le funzionalità del progetto

LIDS

- Riunisce diverse funzionalità in un patch set o modulo LSM:
  - MAC configurabile per mezzo di ACL
  - IDS (rilevazione di port scan)
  - limitazione dei poteri del superutente per mezzo di capabilities
- Esiste per le versioni di kernel 2.2, 2.4, 2.5.
- La configurazione è conservata in quattro file, può essere configurato per consentire l'avvio al boot di una LFS (LIDS Free Session) che non attiva le protezioni che altrimenti impedirebbero la modifica della configurazione.

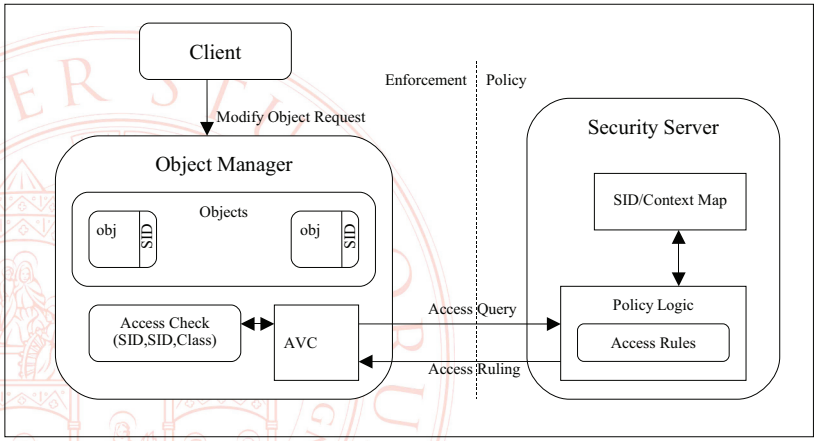
RSBAC

- Uno dei progetti più maturi (nato nel 1996), derivato dal Generalized Framework for Access Control (GFAC) di Marshall Abrams and Leonard LaPadula
- Offre un ampio insieme di funzioni di sicurezza modularmente combinabili, tra cui
  - MAC secondo Bell-LaPadula
  - ACL
  - scansione antivirus trasparente (Dazuko)
  - Role Compatibility
  - Privacy Model (protezione dati secondo le leggi europee)
  - Capabilities

SELinux

- Origini
  - 1973: modello MLS di Bell-LaPadula
  - 1980: MLS influenza fortemente la definizione di TCSEC
  - 1990: NSA e SCC cercano di formalizzare MAC
    - La collaborazione con Utah University porta alla definizione ed implementazione del sistema Flask nel S.O. Fluke
  - 2000: NSA, NA, MITRE lavorano al porting di Flask in Linux, rilasciando nel dicembre SELinux come open source

Architettura Flask



Flask ➡ SELinux

- L'architettura Flask si limita a definire le interfacce che il security server deve esporre agli object manager
- Restano confinati all'interno security server, e sono specifici dell'implementazione
  - L'assegnazione di un eventuale semantica ai SID
  - Il linguaggio di specifica delle policy
    - Nel caso di SELinux le policy risultano dalla combinazione di tre modelli
      - Type Enforcement (TE, obbligatorio)
      - Role-Based Access Control (RBAC, obbligatorio)
      - Multi-Level Security (MLS, facoltativo)

Identificativi di sicurezza

- Ogni soggetto/oggetto è identificato
  - internamente per mezzo di un numero intero chiamato SID (Security Identifier)
  - agli occhi dell'utente per mezzo di una terna chiamata *security context* che contiene attributi immediatamente comprensibili
- Le due viste sono messe in relazione 1:1 tramite una *security context table*
- L'uso dei SID è vantaggioso
  - in termini di efficienza
  - perchè consente di delegare le decisioni di sicurezza senza svelare i dettagli della terna
  - perchè svincola il progetto del security manager dai particolari attributi scelti

Contesto di sicurezza

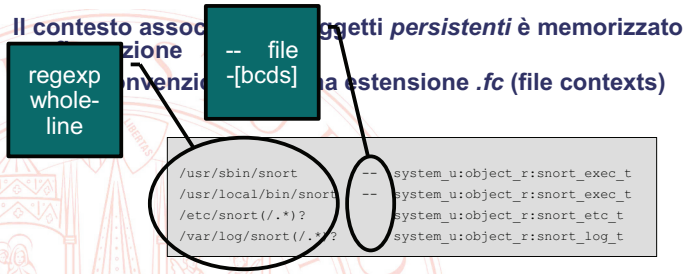
- La semantica del security context o contesto di sicurezza è chiarita dai tre attributi che lo compongono:
  - *User identity*: l'account SELinux associato al soggetto/oggetto.
    - Gli account SELinux sono separati da quelli di sistema.
    - Più account di sistema possono essere mappati su un account SELinux
    - La modifica di account di sistema non invalida le policy SELinux
  - *Role*: il ruolo correntemente utilizzato dall'utente
    - In ogni istante un utente può utilizzare un solo ruolo
    - Il ruolo speciale *sysadm_r* è deputato all'amministrazione del sistema SELinux
    - Il ruolo dummy *object_r* è utilizzato per completare la terna negli oggetti, che solitamente non hanno necessità di un ruolo

Contesto di sicurezza (2)

  - *Type*: il terzo attributo che compone il contesto di sicurezza è quello fondamentale utilizzato da SELinux per prendere le decisioni di autorizzazione,
    - dal punto di vista formale ha due connotazioni ben diverse
      - si parla di *type* per gli oggetti
      - si parla di *domain* per i soggetti
- Per distinguere facilmente fra i tre attributi si usa una convenzione:
  - gli user hanno nomi senza suffisso
  - i role hanno nomi con suffisso _r
  - i type hanno nomi con suffisso _t

Etichettatura

- Il contesto associ… oggetti *persistenti* è memorizzato in un file di …zione
  …nvenzi… …a estensione *.fc* (file contexts)



- I file sono tra gli oggetti più comuni: al fine di velocizzare l'associazione tra un file ed il proprio contesto, viene eseguita un'operazione di *labeling*
  - il contesto viene memorizzato, nei filesystem che li supportano, negli attributi estesi del file

Decisioni di sicurezza

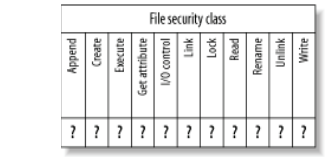
Il security server è chiamato a prendere due tipi di decisione:

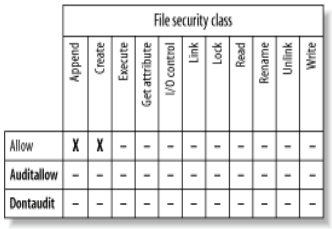
- decisioni di accesso: determinare se un soggetto può o no svolgere una determinata operazione su di un oggetto
- decisioni di transizione: determinare quale tipo assegnare ad un oggetto o soggetto all'atto della creazione

SELinux assume una *closed world policy*, per cui ogni richiesta di autorizzazione è negata dal security server se non esiste una regola che la consente esplicitamente.

Decisioni di accesso

- Ad ogni *classe di oggetti* è associato un *vettore di accesso*, che contiene un bit per ogni azione definita per la classe
- Il security server, interrogato sulla possibilità di effettuare una determinata azione su di un oggetto da parte di un soggetto, restituisce tre particolari istanze del vettore d'accesso, calcolate solo sulla base di
  - dominio del soggetto
  - tipo dell'oggetto
  - classe dell'oggetto





Decisioni di accesso (2)

- La possibilità di effettuare l'azione richiesta dipende dai bit settati nei tre vettori, secondo le regole seguenti:

Bit impostato	Azione permessa	Risultato loggato
nessuno	No	Si
allow settato	Si	No, a meno che non ci sia un'altra regola auditallow
auditallow settato	No, a meno che non ci sia un'altra regola allow	Si
dontaudit settato	No	No

Decisioni di transizione

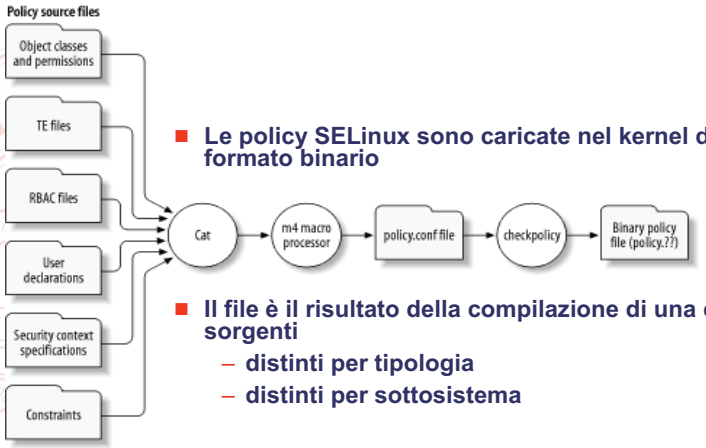
- Riguardano l'assegnamento di un contesto di sicurezza ai nuovi soggetti ed oggetti
- Per i nuovi processi
  - il default è ereditare il contesto del processo che li genera
  - è possibile una transizione ad un altro dominio
    - imposta dal processo padre*
    - richiesta dal processo medesimo*
    - imposta da SELinux in base al tipo del programma eseguito

* Caratteristica fondamentale che distingue SELinux dalle altre soluzioni è la presenza di un'API che consente di scrivere programmi in grado di interagire con il security server, anzichè solamente "subire" una policy

Decisioni di transizione (2)

- Per i nuovi file (come esempio di oggetto)
  - il default è ereditare il contesto della directory che li contiene
  - è possibile una transizione ad un altro tipo
- In ogni caso in cui sia richiesta una transizione, questa deve essere comunque esplicitamente autorizzata dalle policy di sistema

Configurazione



- Le policy SELinux sono caricate nel kernel da un file in formato binario
- Il file è il risultato della compilazione di una collezione di file sorgenti
  - distinti per tipologia
  - distinti per sottosistema

Configurazione

- La struttura del file di configurazione può essere così schematizzata:

```
policy -> flask te_rbac users opt_constraints contexts
```

```
flask -> classes initial_sids access_vectors
te_rbac -> te_rbac_statement | te_rbac te_rbac_statement
te_rbac_statement -> te_statement | rbac_statement
te_statement -> attrib_decl | type_decl |
type_transition_rule | type_change_rule | te_av_rule |
te_assertion | typealias_decl | typeattribute_decl
rbac_statement -> role_decl | role_dominance |
role_allow_rule
users -> user_decl | users user_decl
opt_constraints -> constraints | empty
contexts -> initial_sid_contexts fs_uses
opt_genfs_contexts net_contexts
```

Role-Based Access Control

- Le regole RBAC sono essenzialmente utilizzate per:
  - consentire la mappatura di un utente su di un ruolo
  - consentire l'utilizzo di un dominio ad un ruolo
  - consentire la transizione di un utente da un ruolo ad un altro

```
user_decl -> USER identifier ROLES set ';'
set -> '*' | identifier | '{' identifier_list '}' | '~'
identifier | '~' '{' identifier_list '}'
identifier_list -> identifier | identifier_list identifier
```

```
role_decl -> ROLE identifier TYPES types ';'
types -> set
set -> '*' | identifier | '{' identifier_list '}' | '~'
identifier | '~' '{' identifier_list '}'
```

```
role_allow_rule -> ALLOW current_roles new_roles ';'
current_roles -> set
new_roles -> set
```


Type Enforcement

- Le regole TE sono più numerose e complesse. Permettono di esprimere:
  - la definizione di un tipo, noto con diversi alias e corredato di attributi
    - gli attributi possono essere usati per rendere più chiara la scrittura delle regole di controllo degli accessi, associando un identificatore con semantica evidente ad un dominio. Ad esempio, un tipo definito per etichettare directory utente potrebbe avere l'attributo *home_type*

```
type_decl -> TYPE identifier opt_alias_def opt_attr_list ';'
opt_alias_def -> ALIAS aliases | empty
aliases -> identifier | '{' identifier_list '}'
identifier_list -> identifier | identifier_list identifier
opt_attr_list -> ',' attr_list | empty
attr_list -> identifier | attr_list ',' identifier
```


Type Enforcement (2)

  - le regole che dettano cosa è consentito ai tipi

```
te_av_rule -> av_kind source_types target_types ':' classes permissions
';'
av_kind -> ALLOW | AUDITALLOW | AUDITDENY | DONTAUDIT
source_types -> set
target_types -> set
classes -> set | SELF
permissions -> set
set -> '*' | identifier | nested_id_set | '~' i
nested_id_set | identifier '-'
nested_id_set -> '{' nested_id_list '}'
nested_id_list -> nested_id_element | nested_ _list nested_id_element
nested_id_element -> identifier | '-' ide ier | nested_id_set
```

attributo

  - particolari regole sono le asserzioni *NEVERALLOW*, utili per garantire consistenza

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


Type Enforcement (3)

  - le regole che inducono la transizione di tipo (che deve essere consentita da una regola te_av_rule esplicita)

```
type_transition_rule -> TYPE_TRANSITION source_types target_types ':' classes new_type ';'
source_types -> set
target_types -> set
classes -> set
new_type -> identifier
set -> '*' | identifier | '{' identifier_list '}' | '~' identifier | '~' '{' identifier_list
'}'
```

- Per semplificare la scrittura delle regole di transizione di tipo, esistono 4 macro che generano la corretta combinazione di te_av_rule e type_transition_rule: `domain_auto_trans`, `domain_trans`, `file_type_auto_trans`, `file_type_trans`

Modalità di funzionamento

- Un sistema con SELinux può essere avviato in tre modi:
  - Disabled
    - nessun controllo degli accessi, nessun log, perdita dei SID associati ai file in caso di modifica
  - Permissive mode
    - il controllo degli accessi genera unicamente un log delle decisioni di sicurezza, ma non le mette in atto
    - audit2allow permette di convertire il log in policy TE che permettono l'attuazione delle operazioni registrate (attenzione a effettuare i test in ambiente privo di rischi!)
  - Enforcing mode
    - piena funzionalità del sistema di controllo degli accessi

Un esempio di utilizzo di SELinux:

il confinamento degli utenti che condividono le risorse di un web server

Motivation

Hosting is a widespread solution for the deployment of web sites.

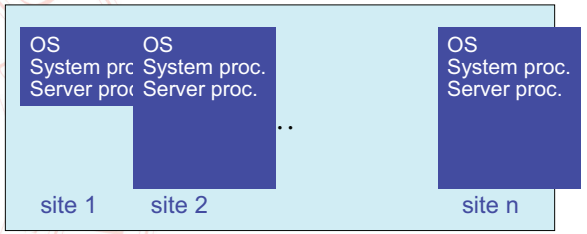
As for many other domains, a satisfactory balance between security and efficiency is not easy to achieve

Desirable features for a multi-site web server include:

- Efficiency
  - share as many resources as possible
- Host protection
  - avoid favoring intrusions through web server faults
- Isolation
  - avoid leakage of confidential data from a site to another

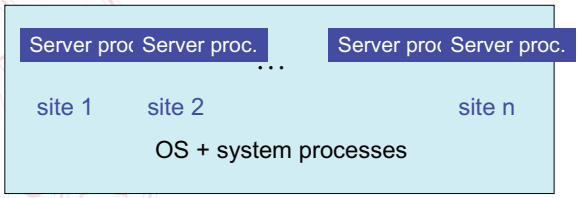
Security vs. efficiency

- Security has many facets - in this context we focus on *isolation*
- Isolation implies sharing few resources - but duplicate resources are wasted
- Several possible solutions offer tradeoffs ranging from the most secure to the most efficient:
  - *Virtual servers*
    - IP sharing
    - System overhead
    - Almost perfect isolation



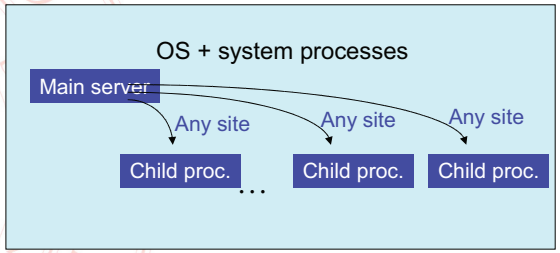
Security vs. efficiency

- Security has many facets - in this context we focus on *isolation*
- Isolation implies sharing few resources - duplicate resources are wasted
- Several possible solutions offer tradeoffs ranging from the most secure to the most efficient:
  - Virtual servers
  - *Virtual hosts on isolated processes*
    - IP sharing
    - Intermediate resource pooling
    - Access to host



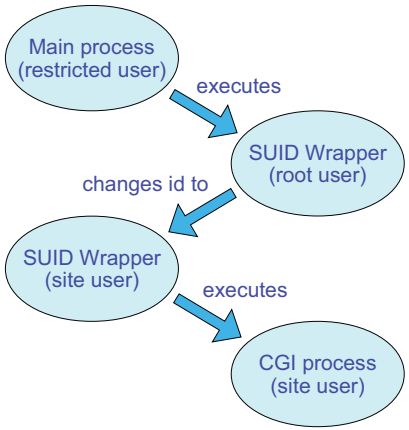
Security vs. efficiency

- Security has many facets - in this context we focus on *isolation*
- Isolation implies sharing few resources - duplicate resources are wasted
- Several possible solutions offer tradeoffs ranging from the most secure to the most efficient:
  - Virtual servers
  - Virtual hosts on isolated processes
  - *Virtual hosts on a single process*
    - Efficient pooling
    - Cross read
    - Access to host



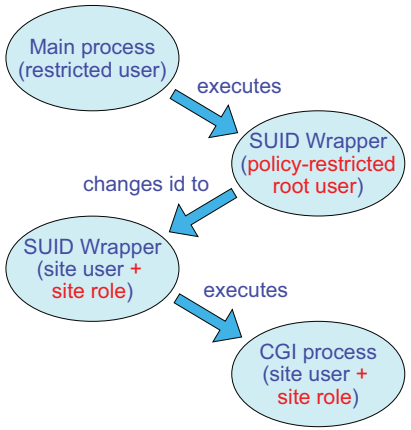
Wrapper-based architectures

- A possible approach to the security/efficiency tradeoff:
  - adopt the single process model
  - do not spawn dedicated children processes for each site
  - standard handling of static content
  - handling of dynamic content delegated to an external program
    - SUID - risky!
    - limited to CGI execution, not suitable for filter-style parsers



Wrapper-based architectures secured

- An *enhanced* approach to the security/efficiency tradeoff:
  - adopt the single process model
  - do not spawn dedicated children processes for each site
  - standard handling of static content
  - handling of dynamic content delegated to an external program
    - MAC limiting root powers
    - still limited to CGI execution, not suitable for filter-style parsers



SELinux

- Implements the security models useful for achieving the outlined features
- Relevant aspects:
  - MLS, UI, RBAC
  - MAC-TE (Mandatory Access Control - Type Enforcement)
    - every object is associated with a security context
    - rules can authorize *domain transitions* for processes
    - rules can *automate* domain assignment

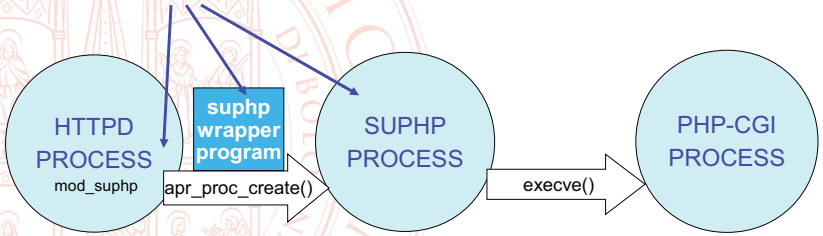
suPHP

- Allows CGI-like processing of PHP pages written for server-side processing
  - can be used for countless applications
- Simple module/wrapper structure
  - easy to modify

→ Good candidate as a testbed for the proposed model

Domain transition

- Recall out goal: achieve security by executing PHP programs within site-specific domain
  - isolation
  - minimum privilege
- Lacking specific configuration, the execution domain is *inherited*
  - Three ways to induce the transition



Common definitions

- File contexts

```
/usr/local/apache2_.*    -d  gen_context(system_u:object_r:apache_suphp_t,s0)
/usr/local/apache2_.*    --  gen_context(system_u:object_r:apache_suphp_t,s0)
/usr/local/apache2_.*/bin/httpd   -- gen_context(system_u:object_r:apache_suphp_exec_t,s0)
```

Common definitions

- Type enforcement

```
type apache_suphp_t;
type apache_suphp_exec_t;

role system_r types apache_suphp_t;

domain_type(apache_suphp_t)
files_type(apache_suphp_t)
domain_auto_trans(unconfined_t,apache_suphp_exec_t,apache_suphp_t);
allow apache_suphp_t apache_suphp_exec_t:file entrypoint;

...

#deve poter leggere i file contenuti nel direttorio /etc
files_read_etc_files(apache_suphp_t)

#Il processo Apache padre è figlio di init e deve poter mandargli un
sigchld quando muore
init_sigchld(apache_suphp_t)
```

```
#permessi per usare le librerie di sistema
libs_use_ld_so(apache_suphp_t)
libs_use_shared_libs(apache_suphp_t)
miscfiles_read_localization(apache_suphp_t)

#permette ad Apache di mandare/ricevere pacchetti tcp su tutte le
interfaccie, verso tutti i nodi e verso tutte le porte
corenet_tcp_sendrecv_all_if(apache_suphp_t)
corenet_tcp_sendrecv_all_nodes(apache_suphp_t)
corenet_tcp_sendrecv_all_ports(apache_suphp_t)
corenet_tcp_bind_http_port(apache_suphp_t)

...

#Il web server apache ha bisogno di numerose capacità, quali poter
effettuare la setuid() e la setgid()
allow apache_suphp_t self:capability { kill net_bind_service setgid
setuid };
...
allow apache_suphp_t self:process { signal signull };
...
```

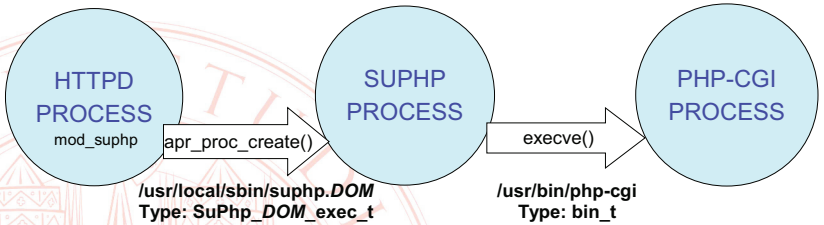

Common definitions

- Template for the site-specific policies

```
#Dichiarazione ruolo
role system_r types SuPhp_<<<Domain>>>_t;

#Dichiarazione Nuovi Tipi
type SuPhp_<<<Domain>>>_t;
type SuPhp_<<<Domain>>>_document_t;

#uso della domain_type e files_type per permettere ai tipi sopra
#dichiarati di essere associati rispettivamente ad un processo e ad
#un file
domain_type(SuPhp_<<<Domain>>>_t)
files_type(SuPhp_<<<Domain>>>_document_t)

...

#SuPhp deve poter usare le librerie condivise..
libs_use_ld_so(SuPhp_<<<Domain>>>_t)
libs_use_shared_libs(SuPhp_<<<Domain>>>_t)
miscfiles_read_localization(SuPhp_<<<Domain>>>_t)
```

```
#direttiva per effettuare il cambio di uid e gid
allow SuPhp_<<<Domain>>>_t self:capability { setgid setuid };

#direttiva per permettere al dominio di Suphp di leggere i file di
#configurazione
files_read_etc_files(SuPhp_<<<Domain>>>_t)

#direttiva per permettere all'eseguibile di poter leggere il proprio
#file di configurazione
allow SuPhp_<<<Domain>>>_t SuPhp_conf_etc_t:file { getattr read };

#direttiva per permettere al dominio di leggere la directory e i file
#associati al sito
allow SuPhp_<<<Domain>>>_t SuPhp_<<<Domain>>>_document_t:dir { search
getattr read };
allow SuPhp_<<<Domain>>>_t SuPhp_<<<Domain>>>_document_t:file
{ getattr ioctl read };

#direttiva per permettere all'eseguibile di effettuare logging
allow SuPhp_<<<Domain>>>_t SuPhp_log_t:file { create append };
allow SuPhp_<<<Domain>>>_t var_log_t:dir { search write add_name };
type_transition SuPhp_<<<Domain>>>_t var_log_t:file SuPhp_log_t;
```


Multi-executable solution



+ driven by SELinux configuration
+ no SELinux-aware suPHP code (module and wrapper) modifications

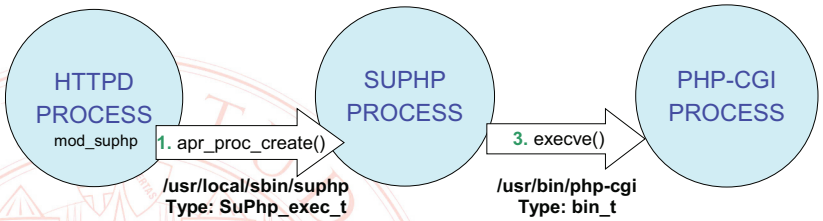
- wrapper file replication
- suPHP modifications to invoke site-specific wrapper

Multi-executable solution

- The `domain_auto_trans` macro tells SELinux to automatically assign the correct domain to the wrapper process
  - checks that the starting domain is Apache's
  - determines the final domain in function of the wrapper's type

```
domain_auto_trans(apache_suphp_t,SuPhp_<<<Domain>>>_exec_t,SuPhp_<<<Domain>>>_t)
allow SuPhp_<<<Domain>>>_t SuPhp_<<<Domain>>>_exec_t:file entrypoint;
```

Wrapper-based solution



+ (?) no SELinux-aware suPHP code modifications in the module
+ single wrapper executable

- small vulnerability window before transition
- more complex SELinux policies
- the site-specific domain is passed as a parameter to the wrapper

Wrapper-based solution

- Slightly more complex SELinux additional configuration
  - concede the right to set process own domain

```
allow apache_suphp_t self:process setcurrent;
```

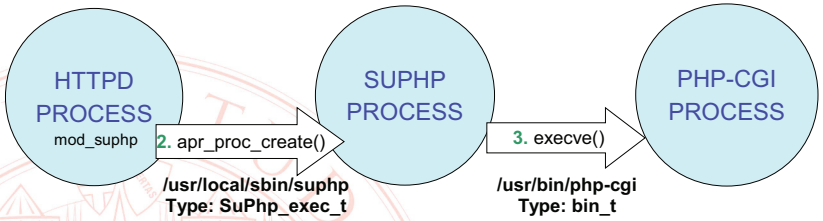
  - concede the right to execute a file of a given type

```
allow apache_suphp_t SuPhp_exec_t:file execute_no_trans;
```

  - concede dynamic domain transition

```
allow apache_suphp_t SuPhp_<<<Domain>>>_t:process dyntransition;
```


Module-based solution



+ simpler policies
+ no vulnerability window for the wrapper
+ no additional parameters passed to the wrapper
+ single wrapper executable

- (?) modifications to the suPHP module

Module-based solution

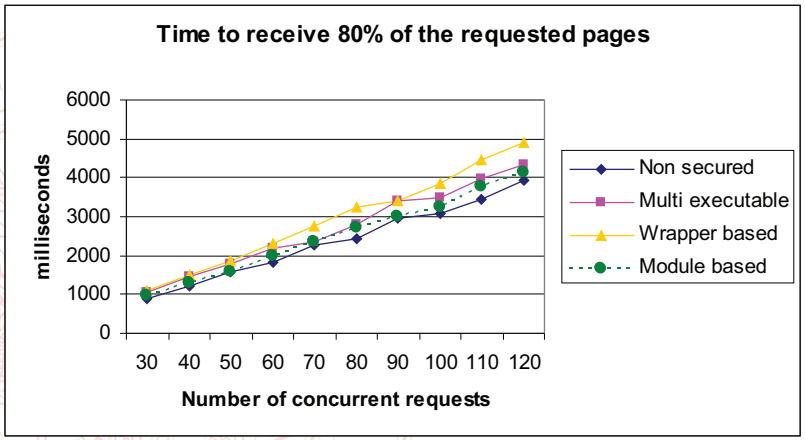
- Very simple additions to the SELinux configuration
  - concede the right to use setexeccon

```
allow apache_suphp_t self:process setexec;
```

  - allow the domain transition

```
domain_trans(apache_suphp_t,SuPhp_exec_t,SuPhp_<<<Domain>>>_t)
```

Performance testing



Riferimenti

- Bill McCarty: "SELinux", O'Reilly, October 2004
- Stephen Smalley, Chris Vance, and Wayne Salamon: "Implementing SELinux as a Linux Security Module", NAI Labs, http://www.nsa.gov/selinux/papers/module-abs.cfm
- Boris Tobotras: "Linux Capabilities FAQ", http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt
- PaX overview - http://pax.grsecurity.net/docs/pax.txt
- Openwall Project - http://www.openwall.com/
- LIDS - http://www.lids.org/
- Leonard La Padula: "Rule Set Modeling of a Trusted Computer System", in "Information Security: An Integrated Collection of Essays", Hrsg.: M. Abrams, S. Jajodia, H. Podell, IEEE Computer Society Press, 1995
- RSBAC - http://www.rsbac.org/