Interceptare apeluri de sistem

Termen de predare: Miercuri 1 Aprilie, ora 23.59 (2009-04-01)

Cuprins

Enunț

Să se scrie un modul de kernel care să monitorizeze apelurile de sistem. La inserare, modulul se va instala în locul unui apel de sistem nefolosit (`MY_SYSCALL_NO`) şi va aştepta următoarele instrucțiuni din userspace:

- `REQUEST_SYSCALL_INTERCEPT` pentru a intercepta un apel de sistem
- `REQUEST_SYSCALL_RELEASE` pentru a renunța la interceptarea unui apel de sistem
- `REQUEST_START_MONITOR` pentru a porni monitorizarea unui apel de sistem (ce a fost în prealabil interceptat) pentru un anumit proces
- `REQUEST_STOP_MONITOR` pentru a opri monitorizarea unui apel de sistem (ce a fost în prealabil interceptat) pentru un anumit proces;

Rutina de tratare a acestor operații va avea urmatoarea signatura:

```
int my_syscall(int cmd, int syscall, int pid);
```

unde **cmd** este comanda aşteptată din userspace, **syscall** este numărul apelului de sistem la care se referă operația, iar **pid** este PID-ul procesului pentru ca se doreşte pornirea sau oprirea monitorizării. Pid-ul este necesar doar pentru operațiile de monitorizare şi va fi ignorat pentru operațiile de interceptare.

Interceptarea unui apel de sistem se referă la faptul că modulul va înlocui intrarea din tabela de apeluri de sistem, astfel încât toate apelurile ulterioare să "trecă prin modul" (şi să ajungă în cele din urmă la apelul de sistem original).

Monitorizarea se refera la faptul că modulul va loga în userspace informații despre procesul şi apelul de sistem: numărul apelului de sistem, parametrii apelului de sistem, codul de eroare întors de apelul de sistem (cel original), pid-ul procesului.

Monitorizarea se poate face pentru un PID sau pentru toate (caz în care se primeşte în loc de pid 0)

Inainte de a efectua operațiile cerute din userspace, modulul va trebui să facă verificări de consistență, drepturi, stare şi să semnaleze, daca este cazul, condițiile de eroare:

- verificări asupra drepturilor [eroare: permisiune/access refuzat]:
    - ♦ doar procesele ce aparțin utilizatorului privilegiat vor putea intercepta sau deintercepta un apel de sistem
    - ♦ doar procesele ce aparțin utilizatorului privilegiat pot porni sau opri monitorizarea unui apel de sistem pentru toate PID-urile
    - ♦ procesele ce aparțin de un utilizator neprivilegiat vor putea opri sau porni monitorizarea unui apel de sistem doar pentru procese sale
- verificări de consistență [eroare: parametru invalid]:
    - ♦ numărul apelului de sistem (se consideră invalid apelul `MY_SYSCALL_NO` sau `__NR_exit_group`)
    - ♦ pid-ul procesului
    - ♦ un apel de sistem ce nu a fost interceptat nu poate fi deinterceptat
    - ♦ nu se poate monitoriza un apel de sistem care nu a fost interceptat
    - ♦ monitorizarea pentru un proces/apel de sistem nu poate fi oprită daca, in prealabil, nu a fost pornita
- verificări de stare [eroare: ocupat]:
    - ♦ un apel de sistem deja interceptat nu poate fi interceptat din nou
    - ♦ monitorizarea nu poate fi pornită de două ori pentru acelaşi apel de sistem si pid
- alte verificări:
    - ♦ modulul trebuie să verifice codul de eroare al funcțiilor apelate şi, în cazul în care nu poate continua, să transmită utilizatorului cauza (exemplu: nu se mai poate aloca memorie)
- cazuri speciale:
    - ♦ modulul trebuie să oprească monitorizarea pentru procesele ce s-au terminat

Se impune folosirea unei liste pentru menținerea informațiilor despre procesele monitorizate. Folosirea unui vector de dimensiune fixa se consideră neadecvată pentru că timpul de căutare va fi similar, complexitatea implementării va fi similară, vectorul va avea un număr limitat de intrări.

Precizări generale

- pentru că numărul de apeluri de sistem este în general mic (250-300) apeluri de sistem, dar şi pentru o latență cât mai mica introdusă de interceptare, se recomandă ca informațiile despre apelurile de sistem interceptate să se țină într-un vector
- pentru interceptarea de apeluri de sistem revedeți informațiile prezentate în Cursul 2

Precizări Linux

- tabela cu apelurile de sistem este exportată pentru module doar în 2.4; pentru 2.6 va trebui sa exportați manual această tabelă, adăugând în fişierul `arch/x86/kernel/i386_ksyms_32.c` următoarele linii:

```
extern void* sys_call_table[];
EXPORT_SYMBOL(sys_call_table);
```

deasemenea va trebuie să modificați `arch/x86/kernel/entry_32.S` pentru a muta tabela din segmentul `.rodata` în segmentul `.data`:

```
.section .data,"a"
```

```
#include "syscall_table_32.S"
```

În plus, pentru că macro-ul `NR_syscalls` nu mai este definit în nucleu, va trebui să definii o variabilă specială care să conină numărul de apeluri de sistem în `arch/x86/kernel/entry_32.S`. Nucleul mainii virtuale definete variabila `my_nr_syscalls` în acest sens.

```
ENTRY(my_nr_syscalls)
.long    nr_syscalls
```

De asemenea, trebuie exportată variabila de mai sus în `arch/x86/kernel/i386_ksyms_32.c`:

```
extern long my_nr_syscalls;
EXPORT_SYMBOL(my_nr_syscalls);
```

imaginea de pe site are facute aceste modificări;

- pentru a folosi în modulul vostru cele două variabile exportate (`sys_call_table` i `my_nr_syscalls`) va trebui să le marcai cu extern:

```
extern void *sys_call_table[];
extern long my_nr_syscalls;
```

- cum kernel-ul 2.6 este preemptiv, trebuie să protejați accesul la datele comune; este indicat să folosiți spinlock-uri
- se impune să se folosească apelul de sistem 0 pentru `my_syscall`, care în prezent nu mai este folosit
- utilizatorul privilegiat va fi `root` (uid 0)
- pentru determinarea uid-ului unui proces folosiți câmpul uid din structura ce identifică procesul (`struct task_struct`)
- `task_struct`-ul procesului curent este dat de macroul **current**
- pentru `task_struct`-ul unui proces oarecare folosiți funcția **find_task_by_vpid(pid)**
- logarea apelului de sistem se va face cu macro-ul **log_syscall** definit în sci_lin.h
- pentru eliminarea proceselor ce s-au terminat va trebui să interceptați apelul de sistem **exit_group**
- codurile de eroare ce trebuie întoarse sunt:
  - `-EINVAL` pentru parametru invalid
  - `-EBUSY` pentru ocupat
  - `-EPERM` pentru access refuzat
  - `-ENOMEM` pentru memorie insuficientă
  - 0 pentru succes

Precizări Windows

- datorită faptului că accesul la tabelele de apeluri de sistem este dependent de versiunea/build-ul sistemului de operare, tema trebuie rezolvată pentru maşina virtuală de pe site; este foarte probabil ca funcțiile helper puse la dispoziție în sci_win.h să nu funcționeze pe alte versiuni

- tabelele de apeluri de sistem pot fi accesate din descriptorii asociați, definiți de

```
struct std KeServiceDescriptorTable[2];
struct std *KeServiceDescriptorTableShadow;
```

structura **std** este definita în sci_win.h

- la inițializarea modulului, va trebui să chemați funcția **get_shadow()** care va inițializa descriptorul pentru tabela shadow; asta pentru că tabela shadow nu este exportată de către kernel

- pentru că nu există apeluri de sitem nefolosite în Windows, va trebui să înlocuiți tabela de apeluri de sistem cu una nouă, în care să adaugați apelul de sistem propriu; atenție, trebuie să înlocuiți tabelele de apeluri de sistem atât în descriptorul principal cât şi în shadow

- numărul apelului de sistem **my_syscall** a fost definit în sci_win.h la **MY_SYSCALL_NO**; puteți presupune că această valoare este întotdeauna mai mare decât numărul de apeluri de sistem prezente

- trebuie tratate doar apelurile de sistem din tabela 0

- pentru determinarea utilizatorului unui process folosiți funcțiile **GetUserOf(pid)** şi **GetCurrentUser**, funcții definite în sci_win.h; pentru a verifica dacă doi utilizatori sunt identici folosiți funcția **CheckUsers**

- utilizatorul privilegiat se consideră utilizatorul cu dreptul de încărcat / scos module din kernel; puteți verifica dacă procesul curent aparține de un utilizator privilegiat cu funcția **UserAdmin**, definită în sci_win.h

- logarea apelului de sistem se va face prin logarea de pachete `struct packet_log` definite în sci_win.h, pachete ce vor fi încapsulate în câmpul **DumpData** al unei intrări de log (vezi **IoWriteErrorLogEntry**) cu codul de eroare 0

- pentru că W2K, WXP, W2k3 sunt preemtive în kernel, trebui să folosiți spinlock-uri pentru a vă proteja de race-uri; folosiți pe cât posibil funcții **Interlocked***

- de la XP se marchează tabela de apeluri de sistem ca read-only; folosiți macrourile **WPOFF** şi **WPON** (definite în sci_win.h) pentru a opri, respectiv porni protecția la scriere

- pentru eliminarea proceselor ce s-au terminat din structurile folosite vedeți **PsSetCreateProcessNotifyRoutine**

- codurile de eroare ce trebuie întoarse sunt:
  - `STATUS_INVALID_PARAMETER` pentru parametru invalid
  - `STATUS_DEVICE_BUSY` pentru ocupat
  - `STATUS_ACCESS_DENIED` pentru access refuzat
  - `STATUS_NO_MEMORY` pentru memorie insuficientă

- ♦ `STATUS_SUCCESS` pentru succes

Testare

Pentru simplificarea procesului de corectare al temelor, dar si pentru a reduce greselile temelor trimise, corectarea temelor se va face automat cu ajutorul unor teste publice (linux, windows). Testele presupun că numele modului de kernel este **sci** atât pe Linux cât şi pe Windows.

Linux

Testul se foloseşte de utilizatorul `nobody` care ar trebuie să existe pe toate sistemele Linux. Din această cauză testul trebuie să fie plasat intr-un director public.

Windows

Testul se foloseşte de utilizatorul `so`. Din această cauză trebuie ca utilizatorul să existe. De asemenea se consideră că utilizatorul are parola `so`. În plus, pentru a putea rula testul, utilizatorul cu care veți rula testul trebuie să aibă drepturile `"Act as part of the operating system"`, `"Create a token object"` şi `"Replace a process level token"`. Pentru a configura aceste drepturi: `Start -> Administrative Tools -> Local Security Policy -> Local Policies -> User Rights Assignments`.

Depunctari

Depunctari pentru probleme constatate in implementarea temei:

- -1.0 warning-uri la compilare
- -0.5 implementare incorecta a cerintelor temei
- -0.2 nu se implementeaza eliminarea proceselor care s-au terminat
- -1.0 nu se folosesc liste in implementare
- -0.1 lipsa readme
- -0.2 lipsa sincronizari sau sincronizare incompleta / incorecta
    - ♦ - 0.1 lipsa sincronizare accces lista procese
    - ♦ - 0.1 lipsa sincronizare vector de apeluri de sistem
- -0.1 utilizare functii la nivele IRQL necorespunzatoare
- -0.1 memory leaks
- -0.1 alte probleme constatate
- -0.0 observatii

In cazuri exceptionale (tema trecere testele prin nerespectarea cerintelor) si in cazul in care tema nu trece toate testele se poate scadea mai mult decat este mentionat mai sus.

Intrebari

Pentru intrebari legate de tema puteti consulta arhivelele listei de discutii sau puteti trimite un e-mail (trebuie sa fiti inregistrati).