Systemrufe

Reinhard Bündgen
buendgen@de.ibm.com

Systemrufe aus libc

- Beispiele
  - E/A: open, close, read, write, poll
  - Prozesse: fork, execve, kill, nice
  - Zeit: time, settimeofday
  - Speicher: mmap, brk
- Beschreibung: man-Seiten (Band 2)
- Schnittstellen:
  - diverse header-Dateien in /usr/include

Schutzmechanismen: Motivation

- Bestimmte »gefährliche« Funktionen sollen vom Nutzer gar nicht oder nur kontrolliert bedienbar sein
  - Funktionen, die HW beschädigen können
  - Funktionen, die das System zum Absturz bringen können
  - Funktionen, die fremde Daten schädigen können
  - Funktionen, die fremde Programme beeinflussen können
- Schutz vor Fehler durch
  - Fehlbedienung
  - Programmfehler
  - Absichtliche Korruption
- Wichtig im Multiuser/Multitasking-Umfeld
- Nutzer sind beliebige Nutzer einschließlich Systemadministratoren (super user)

Schutzmechanismen: Techniken

- Hardwareunterstützung
- Prozessormodi
  - Systemmodus (primary space mode)
  - Nutzermodus (home space mode)
- Unterschiedliche Adressräume
  - kernel space
  - user space
- Software Unterbrechungen
  - Intel: INT 80h oder sysenter (>= Pentium II)
    - (Rückkehr auf Intel: iret oder sysexit Assemblerinstruktionen)
  - System z: SVC

Systemruf: Ablauf

- libc: Systemruf (*Systemrufname*)
- Softwareunterbrechung mit *Systemrufnummer*
  - (gemäß /usr/include/asm/unistd.h)
- linux/arch/architecture/kernel/entry.S: system_call
- Aufruf von Funktion in Systemruftabelle (in entry.S), indiziert durch *Systemrufnummer*:
- sys_*Systemrufname*
- evtl. do_*Systemrufname*

System z Softwareunterbrechung

- SVC (`SVC I [I]`)
  1. speichere aktuelles PSW nach Positionen 320-335 (altes PSW)
  2. lade Positionen 448-463 als neues PSW
  3. speichere SVC-Info nach 138-139
- Rückkehr
  - lade Positionen 320-335 (altes PSW) als aktuelles PSW
  - (Intel: iret oder sysexit Assemblerinstruktionen)

system_call: Resultat und Argumente

- Argumente
  - Systenrufnummer
  - Wertparameter (Intel 4 weitere via Register)
- Resultat:
  - int
  - -1 bis -4095: Fehler
  - wird von libc negiert und in `errno` gespeichert
  - siehe `include/asm-generic/errno[-base].h`
  - >= 0 oder < -4095: Erfolg
- Zeiger: Achtung beim Überschreiten der Adressraumgrenzen.

system_call: Grobstruktur

- Ablegen der system_call Adresse:
  - Intel/int 80h: `trap_init()`: `set_system_gate(SYSCALLVECTOR,&system_call)`
  - s390: `setup_lowcore()`: `lc->svc_new_psw.addr = ...`

- Sichern der Register
- Aufruf, der mit Systemrufnummer assozeierten Funktion
- (evtl. eingerahmt von trace-Routinen)
- Bearbeitung von “bottom halfs”
- evtl: Aufruf des Schedulers
- Bearbeitung von Signalen
- Restaurierung der Register

Datentransfer zwischen user space und kernel space

- `include/asm/uaccess.h`
  - teste Zugriff auf US: `access_ok(type,addr,size)`
  - lese Wert aus US: `get_usr(val, ptr)`
  - schreibe Wert in US: `put_usr(val, ptr)`
  - lese Daten aus US: `copy_from_user(to, from, length)`
    - `__copy_from_user(to,from,length)`
  - schreibe Daten in US: `copy_to_user(to, from, length)`
    - `__copy_to_user(to, from,length)`
  - lese String aus US: `strncpy_from_usr(to, from, length)`
  - messe String in US: `strlen_usr(str)`
  - lösche Speicher im US: `clear_user(mem,len)`

Beispiele für System Calls

- kernel/sys_ni.c: sys_ni_syscall
- kernel/sched.c: sys_nice
- kernel/time.c: sys_time
- kernel/timer.c: sys_sysinfo
- kernel/sys.c: sys_reboot

Virtuelle DSOs

- Problem:
  - Auf x386 hängt es von der Prozessorgeneration ab, ob sysenter unterstützt wird: Kann Linux bestimmen welche Form des Systemrufs benutzt wird?
- Antwort:
  - virtual dynamic shared objects (vdso)
  - eine vom Kern je nach benutztem Prozessortyp in den Programmadressraum eingeblendete dynamische Bibliothek
  - `linux-gate.so.1`