

Übungen zu Systemsicherheit

Jürgen Kleinöder,
Michael Gernoth, Reinhard Tartler
Universität Erlangen-Nürnberg, Informatik 4

E.1 Besprechung der Aufgabe 'token'

- Probleme?

E.2 Systemaufrufe als Bibliotheksfunktionen

- Syscalls sind Betriebssystemspezifisch (Operationen im Kern!)
- Linux implementiert eine POSIX-ähnliche Schnittstelle
- GNU Libc6: Stellt Systemcalls als höhere Bibliotheksaufrufe zur Verfügung (fopen => open, ...)
- In SOS1/Sysprog1 wurden viele Syscalls kennengelernt

E.3 Verfolgung von Bibliotheksaufrufen

- ltrace-Utility

```
>> ltrace ls
__libc_start_main(0x804dfe0, 1, 0xffc73604, 0x80567e0, 0x8056790 <unfinished ...>
isatty(1) = 1
getenv("TABSIZE") = NULL
getopt_long(1, 0xffc73604, "abdcfghiklmnopqrstuvw:XCDEFghi:...", 0x8059580, NULL)
= -1
__errno_location() = 0xf7ce568c
malloc(36) = 0x805cca0
__errno_location() = 0xf7ce568c
malloc(36) = 0x805ccc8
malloc(12800) = 0x805ccf0
malloc(16) = 0x805fef8
strlen(".") = 1
malloc(2) = 0x805ff10
memcpy(0x805ff10, ".", 2) = 0x805ff10
__errno_location() = 0xf7ce568c
opendir(".") = 0xf78c2008
readdir64(0xf78c2008) = 0xf78c2020
readdir64(0xf78c2008) = 0xf78c2038
readdir64(0xf78c2008) = NULL
closedir(0xf78c2008) = 0
__setjmp(0x805bba0, 0xffc730f8, 0xf7dd86f8, 0xf78c2008, 0) = 0
qsort(0x805ccf0, 0, 128, 0x8049f20) = <void>
free(0x805ff10) = <void>
free(NULL) = <void>
free(0x805fef8) = <void>
exit(0 <unfinished ...>
__fpending(0xf7e904e0, 1, 0, 0x8048f9f, 80) = 0
fclose(0xf7e904e0) = 0
```

E.4 Dynamische Bibliotheken

- Sparen Platten und Speicherplatz
- Werden auf 2 Weisen benutzt:
 - ◆ per dynamischem Linker (/lib/ld-linux.so.2)
 - Feste Liste im ELF-Headers des Binaries (objdump -p)
 - Umgebungsvariable LD_PRELOAD
 - ◆ durch Anfragen an den dynamischen Linker während des Programmablaufs

```
void *dlopen(const char *filename, int flag);
void *dlsym(void *handle, const char *symbol);
int dlclose(void *handle);
```

- Linken mit GCC:

```
SOURCES = overlayfs.c utils.c ...

overlayfs.so: $(SOURCES)
$(CC) $(CFLAGS) $^ -o $@ -shared -ldl
```

E.5 Überladen von Bibliotheksaufrufen

- Bibliotheken exportieren Symbole (Funktionen/Variablen), welche von Anwendungen aufgerufen werden können
- Name und Aufrufkonvention in ABI festgelegt
- Der dynamische Linker kann einzelne Symbole überladen
- Umgebungsvariable LD_PRELOAD, fügt Library vor allen anderen hinzu
- Einschränkungen:
 - ◆ nur dynamische Binaries
 - ◆ keine SUID & SGID binaries

E.6 Aufgabenstellung: OverlayFS

- Ziel: Zugriff auf ein zugangsbeschränktes Verzeichnis wird umgelenkt
- Temporäres Verzeichnis wird für den Prozess überlagert
/etc -> /tmp/syssec-10477/etc
- Folgende Funktionen sind mindestens zu überladen:
 - ◆ open, open64
 - ◆ fopen, fopen64
 - ◆ __xstat, __xstat64, __lxstat, __lxstat64
 - ◆ opendir, readdir, readdir64
 - ◆ creat, creat64
 - ◆ access
 - ◆ chmod, fchmod
 - ◆ acl_extended_file

E.7 Hinweise zur Bearbeitung

- 'Original'-Funktionen können per dlsym() adressiert werden:

```
#define _GNU_SOURCE 1
#include <dlfcn.h>

FILE *fopen(const char *path, const char *mode) {
    FILE* (*func) (const char*, const char*) = NULL;

    func = (FILE* (*)(const char*, const char*))
        dlsym(RTLD_NEXT, "fopen");

    return (*func) (path, mode);
}
```

- Es muss nun in 2 Verzeichnissen gesucht werden:
 - ◆ das "eigentliche"
 - ◆ das "temporäre" (/tmp/syssec-UID/echter/Pfad/)
- Es bietet sich an ein shell Wrapper zu schreiben

```
#!/bin/sh
LD_PRELOAD=/path/to/overlayfs.so "$@"
```

E.8 Probleme

■ open{,64}

- ◆ Verwendet variable Anzahl an Argumenten
- ◆ Evtl. vorhandenes Argument muss extrahiert werden:

```
int open (__const char *__file, int flags, ...) {
    mode_t mode = 0;
    va_list args;

    ...

    if (flags & O_CREAT) {
        va_start(args, flags);
        mode = va_arg(args, mode_t);
        va_end(args);
    }
}
```

- ◆ Das echte open immer mit 3 Argumenten aufrufen, auch wenn letztes 0

■ readdir

- ◆ Es wird bei Fehlern der DIR-Pointer des Schattenverzeichnisses benötigt