# Reaktive Sicherheit

Übungsblatt 10

Balduin Binder [s6babind@uni-bonn.de] Charlotte Mädler [s6chmaed@uni-bonn.de]
Bünyamin Sarikaya [s6busari@uni-bonn.de]

Es fehlt: 1.2, 1.3 unclean, 1.5, 3.3, 3.5, 3.6, 3.7 # Aufgabe 1 (1)

```
#include <dlfcn.h>
void *dlopen(const char *filename, int flag);
```

dlopen() lädt die dynamische library die ihm mit dem nullterminierten string filename übergeben wird und returned dessen handle. Falls Filename NULL ist, returned es den handle zum main programm. Ein "/" im Filename wird als Path interpretiert. Sonst wird mittels dynamischer Linker für die library gesucht. Flag muss entweder auf RTLD_LAZY oder RTLD_NOW gesetzt sein. Sie entscheiden ob undefinierte Symbole zwingend resolved werden müssen oder nicht. RTLD_LAZY tut dies nur wenn der code diese referenziert. RTLD_NOW tut dies für alle bevor dlopen() zuende ist.

(2)

Funktionen einer Shared Library können als Konstruktor definiert werden und werden somit als Init-Funkionen direkt beim Laden der Library ausgeführt. Eine solche Function muss folgenden Header haben:

```
static void con() __attribute__((constructor)); // header
void con() {
    printf("I'm a constructor\n");
}
```

(3) PLT und GOT sind zusätzlicher Speicher, genutzt vom Compiler und dynamischen Linker. PLT bedeutet Procedure Linkage Table und wird benutzt um externe funktionen/prozeduren aufzurufen, deren adresse zum zeitpunkt des linkens unbekannt sind. Es wird also für den dynamischen linker überlassen, der das zur Laufzeit resolven soll. GOT bedeutet Global Offsets Table. Jedes mal wo eine Globale Varaible einer shared library vom Programm geladen wird, wird die wahre adresse der Varaible von GOT geladen. Der dynamische linker hat diese vorher gesetzt. Wenn eine Funktion von einer shared library gecalled wird, macht der linker einen jump zu einer adresse im PLT.

(4) The first time the function is called, the PLT code uses offsets stored in the GOT to decide the actual final location of the function, and then: stores this pre-calculated value jumps there The next times the function is called, the value has already been calculated, so it just jumps there directly.

(5)

## Aufgabe 3

(1)Kernelmodule sind codefragmente die nach Bedarf in die Kernel ge- und entladen werden können. Die erweitern die Funktionen der Kernel, ohne ein Systemneustart forauszusetzen. Ein Beispiel wäre der device driver, der dem Kernel erlaubt auf Hardware zuzugreifen welches zum System angeschlossen ist.

lsmod - Zeigt welche Module aktuell geladen sind.

insmod -Fügt ein Modul der Kernel zu. modprobe wird häufiger benutzt, da es mehr abhängigkeiten haben kann.

rmmod - Entfernt Module aus der Kernel.

(2) Programmcode im Ring 0 befindet sich im Kernelmodus.

(3)

(4)printk ist eine C Funktion vom Linux Kernel Interface. Sie druckt Nachrichten zum Kernel Log. Sie akzeptiert ein String Parameter (format sting) der dann geprinted wird. Sie ähnelt printf und Argumente verhalten sich gleich. Sie wird häufig als debugging tool verwendet. Der Grund für die Existenz von printk ist dass die standard C Library in der Kernel mode nicht vorhanden ist.

Loglevel spezifiziert den Nachrichtentyp. Der Syntax ist dabei:

```
printk(KERN_DEBUG "Debug message shown!\n");
```

Log Levels:

0 KERN_EMERG Notfall Condition, System vermutlich tod

1 KERN_ALERT Ein Problem ist aufgetreten, Sofortige Aufmerksamkeit notwendig

2 KERN_CRIT Kritische Condition

3 KERN_ERR Error Aufgetreten

4 KERN_WARNING Warning

5 KERN_NOTICE Normale Nachricht

6 KERN_INFO Information

7 KERN_DEBUG Debug information

Loglevels sind in <linux/kern_levels.h> definiert. Welche log levels geprinted werden ist im /proc/sys/kernel/printk sysctl file festgelegt.

(5)
(6)
(7)

# Aufgabe 4