

OPERATING SYSTEMS UE BONUS BEISPIEL

Vorbereitung

Zur Vorbereitung lesen Sie bitte die ersten drei Kapitel sowie Kapitel 5 des Buches "Linux Device Drivers" [1]. Weiters steht Ihnen ein rudimentäres "Hello World" Modul inkl. Makefile in der Entwicklungsumgebung zur Verfügung. Dieses können Sie gerne als Grundlage für die Entwicklung Ihres eigenen Kernel-Moduls verwenden.

Abgabegespräch

Das Abgabegespräch findet 21.1.2013 (und bei Bedarf am 22. Januar) im TI-Labor statt. Eine Anmeldung im myTI zu einem Slot ist erforderlich - bitte bedenken Sie, dass eine nachträgliche Abgabe **nicht** möglich ist, da die Übung am 25.1. endet. Es gelten die bekannten Richtlinien mit den zusätzlichen Einschränkungen bzw. Lockerungen:

- Ihre Implementierung muss in der TI-Lab Umgebung (User Mode Linux) demonstriert werden.
- GNU C Standard (C99 Standard mit GNU Erweiterungen)
- Kernel Coding Style konform (siehe <kernel source dir>/Documentation/CodingStyle)
- Das Kernel-Modul muss sich sauber (d.h. Freigabe aller verwendeten Ressourcen) aus dem System per `rmmmod(8)` entfernen lassen.
- Definieren Sie den Modul-Parameter `debug`, der, wenn er beim Laden des Moduls auf 1 gesetzt wird, für sinnvolle Debugausgaben (z.B. bei den einzelnen Operationen auf den Character Devices) sorgt.

Secvault, a Secure Vault

Implementieren Sie ein Linux Kernel Modul, welches über Character Devices bis zu 4 flüchtige, aber sichere Speicher (Secure Vaults - kurz: *Secvaults*) zur Verfügung stellt. Ein *Secvault* hat eine `id` und eine konfigurierbare Grösse (`size`) zwischen 1 Byte und 1 MByte. Er soll per eigenem Character Device vom Userspace aus beschreib- und lesbar sein. Daten, die auf ein *Secvault* Character Device geschrieben werden, sollen verschlüsselt als Cyphertext im Speicher abgelegt werden. Beim Lesen soll der im Speicher abgelegte Cyphertext entschlüsselt werden und als Klartext in den Userspace übergeben werden. Als Verschlüsselung soll ein einfaches symmetrisches *XOR*-Verfahren dienen: Abhängig von der Position im *Secvault*, wird ein Schlüsselbyte mit einem Datenbyte per XOR verknüpft:

$$\text{crypt}(\text{pos}, \text{data}, \text{key}) = \text{data}[\text{pos}] \oplus \text{key}[\text{pos} \bmod \text{key_size}]$$

Ein *Secvault* hat also folgende Eigenschaften:

- `id` [0-3]
- `key` (10 Byte)
- `size` [1-1048576]

Die Verwaltung der *Secvaults* soll mit Hilfe des von Ihnen zu entwickelnden Userspace-Tools `svctl` erfolgen:

USAGE: `./svctl [-c <size>|-e|-d] <secvault id>`

Dabei erzeugt die Option `-c` einen neuen *Secvault* der Größe `size` Bytes im Kernel. Weiters sollen bei der Option `-c` von *stdin* 10 Zeichen gelesen werden, die als Schlüssel dienen. Der Rest einer längeren Eingabe wird ignoriert; bei einer kürzeren Eingabe wird der Rest des Schlüssels mit 0x0 gefüllt. Die Option `-e` löscht die Daten eines existierenden *Secvaults* - d.h. der gesamte Speicher wird Kernel-Modul intern mit 0x0 beschrieben (nicht indirekt über das *Secvault* Character Device). Die Option `-d` soll den *Secvault* aus dem System entfernen und den Speicher wieder freigeben. Das Kernel Modul legt direkt nach dem Laden (*insmod(8)*) ein eigenes Character Device (ansprechbar über `/dev/sv_ctl` zur Steuerung und Statusabfrage an. Das Userspace-Tool *svctl* soll per *IOCTL* Calls mit dem Kernel Modul kommunizieren.

Anleitung

- Character (und Block) Devices werden über Major und Minor Device Numbers im Filesystem über *Special Files* referenziert. Nehmen Sie 231 als Major Device Number für das Secvault Device. Die *Special Files* können Sie per *mknod(1)* im Verzeichnis `/tmp/` anlegen (es empfiehlt sich ein Script dafür im Homedirectory anzulegen) oder die vorhandenen Files in `/dev/sv_*` heranziehen.
- Über das *Secvault* Control Device (`/dev/sv_ctl`) können per *open*, *release* und *ioctl* Kommandos an das *Secvault* Device abgesetzt werden.
- Per Userspace-Tool *svctl* sind mit *IOCTL* Calls folgende Operationen möglich:
 - einen Secvault anlegen und dabei Größe und Schlüssel festlegen
 - einen Secvault mit 0x0 initialisieren (= Inhalt löschen)
 - einen Secvault entfernen (inkl. Speicherfreigabe)

Bei der Erstellung eines neuen *Secvaults* soll auch ein neues Character Device (ansprechbar über `/dev/sv_data[0-3]`) angelegt werden.

- Über *Secvault* Data Devices (`/dev/sv_data[0-3]`) soll der Zugriff auf den verschlüsselten Speicher per *open*, *release*, *seek*, *read* und *write* erfolgen.
- Implementieren Sie eine geeignete Behandlung von Fehlerfällen:
 - über Speichergröße des *Secvaults* hinaus lesen/schreiben
 - Anlegen eines existierenden *Secvaults*

Target und Entwicklungsumgebung

Da die Entwicklung von Kernel-Modulen aufgrund der Systemnähe bei Fehlern leicht zum Absturz des kompletten Systems führen kann, haben wir eine Entwicklungsumgebung eingerichtet, die den Normal-Betrieb im TI-Lab nicht stört: Im Verzeichnis:

`/opt/osue/uml/`

befindet sich eine User Mode Linux (UML) Installation. Der darin enthaltene Linux Kernel (*umlkernel*) wird als normaler User Prozess ausgeführt und kann auf dem Host-System auch nur Operationen durchführen, die Sie als Benutzer durchführen können. Insbesondere ist es nicht möglich aufgrund von Programmierfehler das Hostsystem (TI-Lab Client bzw. den TI-Lab Application Server) zum Abstürzen zu bringen.

Im angegebenen Verzeichnis finden Sie außerdem das Script `start`, welches bei Ausführung eine Instanz des UML Kernels bootet. Beim Bootprozess nutzt der UML Kernel ein Minimal-Debian

System¹ (ohne graphischer Benutzeroberfläche) als Root-Image. Bitte ignorieren Sie Fehlermeldungen betreffend des Netzwerkinterfaces. Am Ende des Bootprozesses werden alle virtuellen Konsolen mit 6 Pseudoterminals am Hostsystem verbunden. Die Ausgabe des `start` Skripts bis zum Ende des Bootvorgangs sieht typischerweise so aus:

```
$ /opt/osue/uml/start
Using swap file: /homes/<username>/.uml.swap
Using copy-on-write file: /homes/<username>/.uml.cow
Creating 256M swap file: /homes/<username>/.uml.swap
...
Checking that ptrace can change system call numbers...OK
...
INIT: Entering runlevel: 2
Starting enhanced syslogd: rsyslogd.
Starting periodic command scheduler: crond.
Virtual console 6 assigned device '/dev/pts/17'
Virtual console 5 assigned device '/dev/pts/19'
Virtual console 4 assigned device '/dev/pts/22'
Virtual console 2 assigned device '/dev/pts/24'
Virtual console 1 assigned device '/dev/pts/25'
Virtual console 3 assigned device '/dev/pts/27'
```

Achtung: Das Terminal, in dem Sie die UML Instanz gestartet haben, reagiert nun auf keine Eingaben mehr. Weiters sollten Sie die Grösse des Terminal-Windows nicht verändern. Beachten Sie, dass diese Pseudoterminals je nach Auslastung vergeben werden - d.h. dass Sie i.d.R. nicht immer die gleichen IDs haben. Nun können Sie sich in einem anderen Terminalfenster beispielsweise per `screen` mit einem der Pseudoterminals verbinden (1x Return Taste nach dem Starten von Screen drücken):

```
# screen /dev/pts/25
```

```
Debian GNU/Linux 5.0 sysprog-bonus tty1
```

```
sysprog-bonus login:
```

Der Login lautet (kein Passwort): `root`

Im UML System haben Sie nun root-Rechte und können mit der Entwicklung des Kernel-Moduls beginnen. Wir haben Ihnen dafür noch ein Script geschrieben, welches die Kernelquellen und Ihr Homeverzeichnis in das UML-System einbindet. Führen Sie dazu bitte den Befehl im UML-System aus:

```
~/prepare <ti-lab username>
```

Sie finden danach unter `/usr/src/linux/` die Kernel-Quellen und unter `/root/homedir/` Ihr Homeverzeichnis.

Compilieren des Testmodules

Hier eine kurze Session die das Compilieren, Laden und Entfernen eines "Hello-World" Testmodules zeigt (angenommen wird, dass `~/prepare` bereits aufgerufen wurde):

```
sysprog-bonus:~# ls /usr/src/linux
COPYING      Makefile      block         init          modules.builtin  sound
CREDITS      Module.symvers  crypto        ipc           modules.order    tools
Documentation README         drivers       kernel        net              usr
```

¹<http://www.debian.org>

```

Kbuild          REPORTING-BUGS  firmware  lib        samples      virt
Kconfig         System.map      fs         linux      scripts      vmlinux
MAINTAINERS      arch                include   mm         security     vmlinux.o
sysprog-bonus:~# cd ~
sysprog-bonus:~# cp test_module homedir/ -R
sysprog-bonus:~# cd homedir/test_module
sysprog-bonus:~/homedir/test_module# make clean all
rm -rf Module.symvers *.cmd *.ko *.o *.mod.c .tmp_versions *.order
make V=1 ARCH=um -C /usr/src/linux M=/root/homedir/test_module modules
make[1]: Entering directory '/media/uml/linux-source-2.6.36'
[...]
make[1]: Leaving directory '/media/uml/linux-source-2.6.36'
sysprog-bonus:~/homedir/test_module# insmod ./tm_main.ko
sysprog-bonus:~/homedir/test_module# rmmod ./tm_main.ko
sysprog-bonus:~/homedir/test_module# dmesg | tail
[...]
Hello World! I am a simple tm (test module)!
Bye World! tm unloading...

```

Hinweise

- Entwickeln Sie nach Möglichkeit auf einem TI-Lab Client (d.h. ein tiXX.tilab.tuwien.ac.at Rechner). Diese sind - sofern eingeschaltet - ebenfalls per ssh direkt von außen erreichbar.
- Sie können den Quelltext des Kernelmoduls auch am Host editieren (innerhalb ihres Homeverzeichnis)
- Die UML Instanz können Sie sauber terminieren, indem Sie im UML System den Befehl `halt` ausführen.
- Entwickeln und Testen Sie nach Möglichkeit in kurzen Zyklen: Debugging, wie Sie es im Userspace per `gdb` gewohnt sein mögen, steht Ihnen nur mit sehr viel mehr Aufwand im Kernelmode zur Verfügung. Erweitern Sie daher Ihr Modul in kleinen Schritten und testen gründlich bereits implementierte Funktionen, bevor Sie darauf aufbauen.
- Es können die Character Device Files im Filesystem unabhängig von den tatsächlich vorhandenen Devices im Kernel existieren (d.h. diese müssen nicht direkt vom Modul angelegt werden bzw. wieder gelöscht werden).
- Registrieren Sie gleich beim Laden des Moduls eine *character device region*, in der alle fünf Devices (ein Control- und vier *Secvault*-Devices) Platz finden.
- Sie können das Unixtool `dd(1)` zum Testen der einzelnen Secvaults verwenden. Testen Sie insbesondere Ihre Behandlung der möglichen Fehlerfälle (z.B. Schreiben/Lesen über die Grenzen des Secvaults, ...).
- Mit dem Kommando `su(1)` können Sie die Identität von einem der drei Testuser (test1, test2 oder test3) annehmen.
- Falls die Meldung '`<module name>`' likely not compiled with `-mcmodel=kernel` beim Laden des Moduls via `insmod` auftauchen sollte, bitte UML Instanz neu starten.

Fragen

- Ist es Ihnen möglich den *Secvault* so einzurichten, dass nur diejenigen Benutzer, die ihn angelegt haben, darauf schreiben und davon lesen können? Jeder Benutzer des System soll, sofern der angeforderte *Secvault* frei ist, in der Lage sein einen *Secvault* anzulegen.
Bemerkung: bis zu +5 weitere Bonuspunkte bei korrekter Implementierung
- Wie verhindern Sie unsynchronisierten Zugriff bei "gleichzeitiger" Verwendung des selben *Secvault* Data Devices ?

Literatur

- [1] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005. Verfügbar unter: <http://lwn.net/Kernel/LDD3/>.