



Humboldt University

Computer Science Department
Systems Architecture Group
<http://sar.informatik.hu-berlin.de>

Operating Systems Principles

SecureFS

Zweite Praktikum



4 Wochen Zeit zum Lösen der Aufgaben

Aufgabenstellung auf der SAR Website

Abgabe über GOYA

Abgabefrist: 06.01.2014 09:00

In dieser Woche

Erläutern der Aufgabenstellung

Nächste Wochen

Zeit zur Bearbeitung

Nächste Veranstaltung

11.12.2013

SecureFS



Aufgabe

Implementieren Sie ein Filesystem, bei welchen die Zugriffsrechte nicht nur durch User- & Group-ID festgelegt ist, sondern auch durch die zugreifende Applikation. Es soll zudem möglich sein Dateien und Verzeichnisse auch vollständig vor einem Nutzer zu verbergen.

Zur Konfiguration der Zugriffsrechte soll eine virtuelle Datei dienen. Über eine weitere Datei sollen ausserdem Zugriffstatistiken zur Verfügung gestellt werden.

Nutzen Sie für die Implementierung FUSE und erweitern Sie das entsprechende Beispiel „securefs“, welches auf „fusexmp“ basiert.

SecureFS



Vorgabe:

- securefs (fusexmp)
- makefile
- Testskript
- Test-Dateien/-Verzeichnisse
- Readlink-Beispiel

SecureFS - Details



makefile (Ziele):

- all (default): baut das Userspace-Programm
- mount: mountet „/“ unter mount_securefs
- umount: hängt das Dateisystem wieder aus
- test: baut, mountet und startet test.sh

get_symlink.c

- Aufruf: get_symlink symlink
- Ausgabe: Ziel des Symlinks

securefs.c

- Implementierung von FUSExmp
- Funktionen müssen entsprechend erweitert werden

SecureFS - Details



test.sh

- Testskript

perm.example

- Beispiel für Eingabe (Rechtdatei)

stats.example

- Beispielausgabe von stats

securefs_test

- Getestete Verzeichnisstruktur

SecureFS - Details



Virtuelle Datei „stats“

- Virtuelle Datei unterhalb von „/“
- Lesbar für alle
- Anzahl der Lese-/Schreibzugriffe auf eine Datei durch einen Nutzer mit einer Anwendung
- Absoluter Pfad von Datei- und Programmnamen
- User: numerische UserID (z.B. root -> 0)
- Bsp. (erste Zeile (Spaltenname) ist optional):

file	application	user	read	write
/tmp/securefs_test/file_a	/bin/cat	1000	10	0
/tmp/securefs_test/file_a	/bin/tee	1000	0	2

SecureFS - Details



Hinweis:

- Absoluter Pfad des Programms zu einer ProzessID mittels readlink
- /proc/pid/exe ist symbolischer Link auf Programm
- **get_symlink.c**

```
int main(int argc, char* argv[]) {
    char buf[512];
    int count = readlink(argv[1], buf, sizeof(buf));
    if (count >= 0) {
        buf[count] = '\\0';
        printf("%s -> %s\\n", argv[1], buf);
    }
    return 0;
}
```

SecureFS - Details



perm

- Virtuelle Datei unterhalb von „/“
- Schreibbar für alle
- Setzt Zugriffsrechte (Flags) für Dateien in Abhängigkeit vom Nutzer und der Anwendung
- Für Dateien, die nicht enthalten sind, gelten die Zugriffsrechte des Unix-Dateisystems
- „Zugriffsrechte“:
 - r – readable: Datei/Verzeichnis ist lesbar, aber nicht schreibbar
 - s – shown: Datei/Verzeichnis wird angezeigt (z.B. mit ls) ist aber nicht lesbar/schreibbar
 - h – hidden: Datei/Verzeichnis wird nicht angezeigt, nicht les-/schreibbar
- Allgemein:

file	application	user	mode
------	-------------	------	------

- Beispiel:

/tmp/securefs_test/file_a	/bin/cat	1000	r
9 /tmp/securefs_test/file_a	/bin/tee	1000	h



SecureFS - Details



perm

- Setzen der Rechte erfolgt mittels Schreiben in perm
- Erneutes Schreiben in „perm“ überschreibt alte Rechte
- Beispiel: `cat permissions.txt > ./mount_securefs/perm`

SecureFS - Details



Beispiel für „ls“

- „ls ./mount_securefs/“:

```
bin boot dev ...
```

```
etc home initrd.img ...
```

```
lib lost+found mnt opt ...
```

perm

```
proc root run ...
```

```
sbin selinux srv ...
```

stats

```
sys tmp usr var vmlinuz
```