

Security Engineering

2. Übung

Aufgabe 1 (Links und tar)

In dieser Aufgabe benötigen Sie evtl. Informationen aus den Manualpages von `ln` und `tar`.

Erzeugen Sie mit dem Kommando `dd` eine Datei, die 1 MB groß ist und zufällige Bits enthält:

```
dd if=/dev/urandom of=testfile bs=1M count=1
```

Erzeugen Sie einen Hardlink `testlink` auf `testfile`.

Archivieren Sie beide Dateien mit `tar`. Wie groß ist das Archiv?

Ändert sich die Antwort, wenn `testlink` ein Softlink (engl. *symbolic link*) ist?

Aufgabe 2 (Systemcalls)

Tragen Sie für jedes der folgenden Kommandos folgende Angaben zusammen:

`rm`, `mv`, `chmod`, `chown`, `mkdir`, `rmdir`, `kill`, `ln`, `sleep`, `wget`

(evtl muss `wget` auf Ihrem Rechner nachinstalliert werden)

- a) eine Kurzbeschreibung (eine Zeile, siehe Manualpage)
- b) den Installationspfad
 - mit Hilfe des Kommandos `which`
 - Programme in `/bin` und `/usr/bin` zählen zur Basisinstallation des Betriebssystems
- c) ein Beispiel
- d) den/die entscheidenden Systemcall(s) (höchstens 2)
 - die Systemcalls können wie in der Vorlesung dargestellt mit den folgenden Kommandos ausgegeben werden \leadsto Manualpage lesen
 - `ktrace` und `kdump` (BSD)
 - `strace` (Linux)

- die entscheidenden Systemcalls tauchen relativ am Ende der Ausgabe auf, weil vorher Libraries geladen und Nebenbedingungen überprüft werden
- oft heißt der Systemcall genau so wie das Kommando
- Bei **wget** interessiert hauptsächlich, an welcher Stelle die Verbindung zur Webseite aufgebaut wird.

Beispiel: Kommando *rm*

- a) löscht Dateien, aber standardmäßig keine Directories
- b) `/bin/rm`
- c) `rm text.doc`
- d) `unlink()`

Aufgabe 3 (Inode Informationen)

Sehen Sie sich die Manualpage `stat(2)` an.

Diese beschreibt drei Funktionen, beachten Sie deren Unterschiede. Benutzen Sie diejenige Funktion, die für einen symbolischen Link den Dateityp *symbolischer Link* ausgibt und nicht den Dateityp, auf den durch den Link verwiesen wird.

Schreiben Sie nun ein C-Programm, das für beliebig viele als Kommandozeilenparameter angegebene Dateien (falls diese existieren) die im folgenden genannten Angaben ausgibt.

- Filetyp
- User-ID und Gruppen-ID (Besitzer der Datei, Gruppeneigentümer) und den Namen des Benutzers (`getpwuid()`).
- Zugriffsbits im Oktalsystem
- Zeit des letzten Zugriffs
- Zeit der letzten Inode-Änderung
- Zeit der letzten Dateiänderung
- Zeit der Dateierstellung

Das Ausgabeformat der Uhrzeit sollte der Ausgabe des `date` Kommandos der Shell entsprechen (vgl Übung 1).

Welche Filetypen gibt Ihr Programm für die folgenden Dateien aus (pipe=FIFO):

Teil	Datei	regulär (a)	dir (b)	pipe (c)	socket (d)	char (e)	link (f)
a)	/dev/random						
b)	/bin/sh						
c)	/usr/bin/tar						
d)	/var/spool						
e)	/etc/services						
f)	/tmp/.X11-unix/X0						

Hinweis: Sie können die Ausgabe Ihres Programms mit der des `file`-Kommandos vergleichen.

Falls die angegebene Datei nicht vorhanden ist, soll eine Fehlermeldung ausgegeben werden (`perror()` benutzen, um `errno` auszuwerten).

Erzeugen Sie ein `Makefile`, damit Ihr Programm mit dem Kommando `make` übersetzt werden kann.

Aufgabe 4 (Message Authentication Codes)

Ein Message Authentication Code beweist die Echtheit einer Nachricht zwischen Kommunikationspartnern, die den gleichen geheimen Schlüssel kennen.

Beispiel: wenn Alice an Bob die Datei `/etc/services` sendet und den Hexcode `35423a2579ff41daef9e839a77d88117dc14ba0e9f8a37c1b762585f673a7cc4` hinzufügt, dann weiß Bob

- die Datei wurde nicht verändert
- die Datei stammt von Alice

sofern er den geheimen Schlüssel von Alice und die verwendete MAC-Methode kennt (in diesem Fall den 128-Bit Schlüssel `0123456789abcdef8877665544332211`). Die Standard-MAC-Methode ist der Hashmac oder kurz HMAC. Spielen Sie die Rolle von Bob und führen folgendes Kommando aus.

```
openssl dgst -sha256 -mac HMAC -macopt \
hexkey:0123456789abcdef8877665544332211 /etc/services
```

Hinweis: ein `\` am Ende einer Zeile setzt die Eingabe in der nächsten Zeile fort.

Erzeugen Sie einen zufälligen 128-Bit Kryptoschlüssel mit Hilfe des `/dev/random` Device und damit einen entsprechenden HMAC für die `/etc/services` Datei. Hierfür ergründen Sie die Wirkungsweise der folgenden Kommandosequenz

```
od -t x4 /dev/urandom | head -1 | cut -c 17- | sed -e "s/ //g"
```

Beschreiben Sie kurz in Stichworten, was bei der Kommandosequenz passiert.

Unter manchen Linux Distributionen bekommen Sie damit nicht 128 Bits. Verändern Sie den Parameter hinter `cut` entsprechend, dass es 128 Bits werden.