Betriebs- und Kommunikationssysteme

WS 14/15

Prof. Dr-Ing. Volker Roth

Freie Universität Berlin

3. Übung

Ausgabe	Abgabe
31.10.14	14.11.14

Bitte bei der Abgabe Name der Mitglieder einer Gruppe, Nummer der Übung/Teilaufgabe und Datum auf den Lösungsblättern nicht vergessen! Darauf achten, dass die Lösungen beim richtigen Tutor abgegeben werden. Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind. Nutzen Sie dazu die Flags `-std=c99`, `-Wall` und `-pedantic`. Es sollten keine Warnungen auftauchen.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Prozesse (2 Punkte)

In dieser Aufgabe sollen Sie sich genauer mit dem *Process Control Block* (PCB) befassen und für die Sicherheit in Betriebessystemen mit Hilfe eines Fallbeispiels sensibilisiert werden. Ziel ist es `nicht`, dass Sie den im Artikel beschrieben Angriff reproduzieren können oder die Sicherheitslücke im kleinsten Detail verstehen. Sie sollten vor der Bearbeitung die Hinweise lesen.

(a) Grenzen Sie *Process* und *Thread* voneinander ab.

(b) Beschreiben Sie die Funktionen der Felder *Process identifiers*, *CPU state*, *Control information* und erklären Sie die Notwendigkeit dieser Felder.

(c) Erklären Sie den *Program Counter* und *Stack Pointer*.

(d) Recherchieren Sie den Begriff *Call Stack* (Aufrufstapel) und beschreiben Sie diesen. Was ist eine Rücksprungadresse und warum ist die notwendig?

(e) Lesen Sie den verlinkten Artikel [One98]und beantworten Sie folgende Fragen:

(1) Welche Datenstruktur in C soll ausgenutzt werden?
(2) Was passiert im Codebeispiel `example2.c`? Gegeben Sie ggf. eine Zeichnung an.
(3) Welches Sicherheitsproblem wird durch das Codebeispiel `example2.c` deutlich?
(4) Was kann der Programmierer dagegen beim Programmieren tun?
(5) Welchen Wert auf dem *Stack* möchte der Angreifer manipulieren und warum?
(6) Was möchte der Angreifer erreichen und warum ist es lohnenswert?

Bonusfragen:

(g) Warum muss der *Shellcode* in Assemblersprache übersetzt werden?
(h) Warum kann der Anfang des *Stacks* leicht ermittelt werden?
(i) Es ist schwer die genaue Rücksprungadresse zu finden. Wie kann dieses Problem umgangen werden?

(f) Fassen Sie den Angriff *Buffer overflow* zusammen.

(g) Wie kann ein Betriebssystem einen *Buffer overflow* verhindern?

Hinweise:

- Ein guter Startpunkt zur Recherche ist die Einführung des verlinkten Artikels.
- Die Codebeispiele sind für die Beantwortung der Fragen hilfreich aber nicht notwendig.
- Für die Fragen 1 bis 6 der Teilaufgabe e reicht es aus die ersten sechs Seiten zu verstehen.

Aufgabe 2: C Programmierung (3 Punkte)

a) Wählen sie sich eine shell (bash, csh, etc.) und machen Sie sich mit den Grundlagen, wie z.B. Dateien erzeugen, kopieren, verschieben, Anzeigen von Dateiinhalten und Auflisten von Ordnerinhalten, vertraut.

b) Machen Sie sich mit Low-Level-IO Systemaufrufen unter Linux vertraut. Für diesen Teil der Aufgabe benötigen Sie nur `open()`, `close()`, `read()` und `write()`. Verwenden sie **nicht** die Standardbibliotheksfunktionen für IO! Schreiben Sie eine Funktion

```
int copy(char *sourcename, char *targetname);
```

Die Funktion soll den Inhalt einer Datei kopieren und zwar nur dann, wenn es noch keine Datei mit dem Namen der Zieldatei existiert. Eine Datei darf nicht einfach überschrieben werden. Verwenden Sie zum kopieren einen Puffer (char buffer[BUFFSIZE]). (Experimentieren Sie wenn sie wollen mit unterschiedlichen Puffergrößen und vergleichen sie die benötigte Zeit.)

c) Verwenden Sie Ihre copy-Funktion um einen einfachen Papierkorb zu implementieren. Der Papierkorb soll drei Befehle unterstützen:

```
./trashcan -p filename
```

PUT: Eine Datei innerhalb des Verzeichnisses in dem trashcan ausgeführt wird, soll in einen versteckten Ordner `".ti3_trash"` verschoben werden.

```
./trashcan -g filename
```

GET: Eine Datei innerhalb von `".ti3_trash"` soll wiederhergestellt werden, im selben Verzeichnis, in dem trashcan ausgeführt wird.

```
./trashcan -r filename
```

REMOVE: Die Datei filename in dem Ordner `".ti3_trash"` wird endgültig gelöscht.

Das Verzeichnis soll automatisch beim ersten Aufruf von `trashcan` erzeugt werden. Finden Sie zum Erstellen eines Ordners sowie für das Löschen von Dateien entsprechende Systemaufrufe. In keinem Fall darf eine existierende Datei überschrieben werden. Überlegen Sie sich sinnvolle Fehlerbehandlungen. Sie können das Grundgerüst `trashcan_framework.c` von der Veranstaltungsseite verwenden.

Literatur

[One98] Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 49(14), 1998.