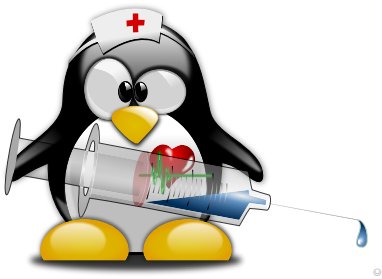
Fachprojekt „Systemsoftwaretechnik“
A1 – Debugging im Linux-Kern

1 Debuging im Linux-Kern - Teil 1



In dieser Aufgabe befasst Ihr Euch mit den eingebauten Werkzeugen zur Fehlersuche in Linux. Hierzu erhaltet Ihr von uns einen präparierten Linux-Kern, den Ihr passend übersetzen müsst. Ihr befasst Euch mit den in der Übung vorgestellten Werkzeugen und lernt sie anzuwenden.

Ladet Euch hierzu mittels `git` den Quellcode für die Aufgaben aus der Vorgabe[1] herunter. Achtet auf die Hinweise am Ende des Übungsblatts.

1.1 Fehler finden

Für diese Teilaufgabe benötigt Ihr die Kernellaufzeitparameter zur Ausnahmebehandlung – siehe Übungsfolien. Wechselt in dem Linux-Repository in den Zweig `a1.1` – siehe `git checkout`. Nutzt die vorhandene Konfigurationdatei `a1.1.config`. Diese müsst Ihr nach `.config` kopieren. Übersetzt nun den Kern, startet Eure virtuelle Maschine und beobachtet das Verhalten des Kerns. Schaut Euch von Zeit zu Zeit die Ausgabe des Kern-Logs mittels `dmesg` an. Probiert außerdem aus, ob das Kern-Log noch Eingaben entgegennimmt: `echo XX > /dev/kmsg`. Schreibt einen Text wie z. B. „test“ in das Log. Probiert dies auch mehrfach aus.

Nutzt nun die vorgestellten Werkzeuge zum Thema Ausnahmebehandlung, um das Verhalten näher zu ergründen. Haltet in der Textdatei `a1.txt` Eure Beobachtungen sowie etwaige Meldungen des Kerns fest. Ergründet anhand des Quellcodes, wie der Fehler funktioniert, und notiert Eure Erkenntnisse.

1.2 Speicherlecks

In dieser Aufgabe befasst Ihr Euch mit dem Auffinden von Speicherlecks. Nutzt die vorhandene Konfigurationdatei `a1.2.config`. Informiert Euch anhand der Dokumentation zu dem Werkzeug `Kernel Memory Leak Detector`, welche Konfigurationsoptionen gesetzt werden müssen, und aktiviert sie. Übersetzt jetzt den Kern und startet die virtuelle Maschine. Stellt hierzu dem Universum (`/dev/the-universe`) eine Frage und empfangt die Antwort des Universums:

```
sst@sst: ~$ echo -n "A question?" > /dev/the-universe
sst@sst: ~$ cat /dev/the-universe
```

Untersucht den Kern auf Speicherlecks. Führt die Schritte auch mehrfach durch. Es kann Euch helfen, wenn Ihr Euch jetzt schon mit dem Werkzeug `Dynamic Debug` – siehe unten unter 1. – befasst.

Wenn Ihr ein Speicherleck gefunden habt, behebt dieses, indem Ihr den Quellcode anpasst, neu übersetzt und erneut testet. Haltet Eure Änderungen anschließend in einem Patch `fix-1.2.patch` in Eurem Aufgaben-Repository fest. Ihr könnt z. B. den Befehl `git diff > fix-1.2.patch` dazu nutzen.

Notiert zusätzlich in der Textdatei `a1.txt` folgende Informationen:

1. Schaut Euch die Dokumentation zu `Dynamic Debug` an.
   a) Wie könnt Ihr die Debug-Ausgaben für unser `sst_chrdev`-Modul bereits beim Start des Kerns aktivieren?
   b) Wie aktiviert Ihr zwei beliebige Zeilen Debug-Ausgabe aus unserem `sst_chrdev`-Modul?
2. Wie funktioniert der Algorithmus zur Detektion von Speicherlecks?

[1] https://git.cs.tu-dortmund.de/sst-ss24/linux

3. Wie sorgt man dafür, dass der Detektor standardsmäßig deaktiviert ist?
4. Für diese Teilaufgabe bitte den Tutor fragen. ☺

1.3 Unerlaubte Zugriffe

Nun sollt Ihr Euch mit unerlaubten Speicherzugriffen im Linux-Kern befassen. Nutzt die vorhandene Konfigurationdatei `a1.3.config`. Nutzt hierzu analog zu Aufgabe 1.2 den Zweig `a1.3`. Auch hier müsst Ihr zunächst das Werkzeug `KASAN` in der Konfiguration aktiveren und Euren Kern übersetzen. Bittet erneut das Universum um Hilfe!

Wenn Ihr einen unerlaubten Speicherzugriff gefunden habt, behebt diesen, indem Ihr den Quellcode anpasst, neu übersetzt und erneut testet. Haltet Eure Änderungen anschließend in der Datei `fix-1.3.patch` in Eurem Aufgaben-Repository fest. Notiert zusätzlich in der Textdatei `a1.txt` folgende Informationen:

1. Erklärt den Unterscheid zwischen beiden Instrumentierungsvarianten `inline` und `outline`?
2. Welche Kontextinformationen liefert Euch der Kern im Kern-Log bei einem Zugriffsfehler? (Es ist keine vollständige Liste erforderlich.)

1.4 Synchronisationsfehler

Die Aufgabe widmet sich dem Werkzeug `locking rule validator` im Linux-Kern. Ihr sollt Synchronisationsfehler im Linux-Kern finden. Nutzt die vorhandene Konfigurationdatei `a1.4.config` wie auch den Zweig `a1.4` und aktiviert in der Konfiguration das Werkzeug `lockdep`. Hofft auch hier wieder auf die Hilfe des Universums, um die Fehler auszulösen.

Begebt Euch nun auf die Suche nach den Ursachen, behebt diese, indem Ihr den Quellcode anpasst, neu übersetzt und erneut testet. Haltet Eure Änderungen anschließend in der Datei `fix-1.4.patch` in Eurem Aufgaben-Repository fest. Notiert zusätzlich in der Textdatei `a1.txt` folgende Informationen:

1. Was beudetet der Zustand `hardirq-safe`?
2. Mit welcher Funktion bzw. Makro kann ein(e) Programmier:in zur Laufzeit eine Ausnahme bzw. Warnung auslösen, wenn eine Sperre wider Erwaten nicht gehalten wird?

Allgemeine Hinweise:

- Mit diesen beiden Kommandos könnt Ihr Euch zügig eine neue Konfiguration erstellen: `make x86_64_defconfig` und `make kvm_guest.config`
- Falls der Bildschirm nicht ausreicht, um eine Fehlermeldung vollständig zu betrachten, könnt Ihr die serielle Konsole nutzen. Hierzu müsst Ihr dem Linux-Kern einen zusätzlichen Parameter übergeben[a]. Schaut Euch neben der Dokumentation auch die Hilfeausgabe des Skripts `boot.sh` an. Diese zeigt Euch, wie Ihr Parameter an den Kern übergeben könnt.
- Achtet darauf, dass Ihr das Pseuo-Dateisystem `debugfs` aktiviert und eingehangen habt. Letzteres erreicht Ihr mit dem Befehl: `mount -t debugfs none /sys/kernel/debug`.
- Das `boot.sh`-Skript leitet den Netzwerkport 22 an den Port `4711` des Hosts weiter. So könnt Ihr Euch, sofern SSH installiert ist, via SSH in der `VM` anmelden. Ihr müsst dem `ssh`-Kommando nur die richtigen Parameter Port angeben – siehe `man ssh`. Verwendet als Host `localhost`. Wenn Ihr außerdem in `/etc/ssh/sshd_config` die Variable `PermitRootLogin` einkommentiert und auf `yes` setzt, könnt Ihr Euch direkt als `root` per SSH anmelden.

[a] https://www.kernel.org/doc/html/latest/admin-guide/serial-console.html