Studienprogramm Sachbearbeiter:in Digitale Forensik
Praktikum Digitale Forensik Grundlagen
Dozenten: Tim Wetterau | Leander Hossfeld
wetterau@hs-mittweida.de | hossfeld@hs-mittweida.de
Stand: 03.04.2023

HOCHSCHULE MITTWEIDA
University of Applied Sciences

Digitale Forensik

Teil 4 – Shell-Skripting

In diesem Praktikum sollen die Grundlagen der Shell-Programmierung an drei Aufgaben verdeutlicht werden. Stellen Sie sicher, dass Sie die grundlegenden Kommandos für die Kommando-basierte Anwendung von Linux beherrschen und die Ansätze zur Shell-Programmierung in der Vorlesung verstanden haben.
Die Voraussetzung für ein erfolgreiches Praktikum ist die Ausführung der Praktikumsaufgaben in der Bourne Again Shell (BASH). Aus der Vorlesung sollte bekannt sein, dass es in dieser Shell zu Problemen mit deutschen Umlauten kommen kann. Daher sind diese zu vermeiden.
Ein Shell-Skript beschreibt den Ablauf bestimmter Kommandos. Daher eignet sich ein Skript sehr gut zu forensischen Analysen, die stets gleich sind. Beispielsweise, dass auslesen von registrierten Nutzern, Programmversionen oder der Sicherung spezieller Dateien. Durch das Skript werden diese nur einmal in Textform in einer Datei gespeichert und können anschließend durch das Aufrufen der Datei ausgeführt werden.

Vorbereitung

Zur Durchführung des Praktikums benötigen Sie die Kenntnisse aus den Praktika 1 bis 3 sowie eine funktionierende Linux Mint-VM. Stellen Sie zunächst sicher, dass diese startet und Sie ein Terminal geöffnet haben.

Erstellung eines Shell-Skripts

Öffnen Sie ein Terminal und schauen Sie sich die Ausgabe der folgenden Befehle an:

```
$ uname -a
$ cat /etc/passwd
$ cat /etc/group
$ cat /proc/cpuinfo
$ df -h
```

Diese Befehle sollten Ihnen bekannt sein und beispielhaft zeigen, welche Standardinformationen aus einem Linux-System gewonnen werden können. Da wir diese Informationen immer von einem System sichern wollen, bauen wir uns daraus ein Skript zur Arbeitserleichterung in der Zukunft.
Wir benutzten zum Bearbeiten den nano-Editor. Legen Sie das script an mit:

```
$ nano basic_info.sh
```

Die Informationen speichern wir in entsprechenden Dateien in einem neuen Unterordner ab. Dazu beginnen wir das Skript mit:

```
mkdir main_info
cd main_info
```

Nun speichern wir die Informationen aus den oben getesteten Befehlen in dem soeben erstellten Ordner ab. Dazu ergänzen wir die folgenden Befehle:

```
uname -a > os_info
cp /etc/passwd passwd
cp /etc/group group
cat /proc/cpuinfo > cpu_info
df -h > storage_info
```

Speichern Sie die Datei mit **Strg+o** und beenden Sie den Editor mit **Strg+x**. Starten Sie anschließend das Skript mit:

```
$ bash ./basic_info.sh
```

Gerne können Sie nun in den Ordner „**main_info**“ wechseln und sich die erzeugten Dateien anschauen. Um das Betriebssystem zu informieren, dass es sich bei dieser Datei um ein ausführbares Programm handelt, ändern wir die Rechte. Dafür geben Sie folgenden Befehl ein:

```
$ chmod u+x basic_info.sh
```

Anschließend können wir in der ersten Zeile im Skript angeben, mit welchen Interpreter das Skript auszuführen ist. Öffnen Sie dafür das Skript erneut mit:

```
$ nano basic_info.sh
```

und ergänzen Sie in der ersten Zeile:

```
#!/bin/bash
```

Nun können Sie das Skript starten mit:

```
$ ./basic_info.sh
```

Sie merken, dass das Skript meckert, da der Ordner „**main_info**“ bereits besteht. Damit das Erstellen des Ordners nur ausgeführt wird, wenn dieser **nicht existiert**, prüfen wir dessen Existenz mit einer Abfrage. Schreiben Sie folgende Abfrage unter die erste Zeile:

```
if [ ! -d main_info ]
then
        mkdir main_info
        cd main_info
        uname -a > os_info
        cp /etc/passwd passwd
        cp /etc/group group
        cat /proc/cpuinfo > cpu_info
        df -h > storage_info
fi
```

Derzeit werden Daten in dem Ordner main_info bei jeder Ausführung überschrieben. Das ist aus forensischer Betrachtung eine Katastrophe. Es könnten Spuren zerstört werden, falls es auf dem untersuchten System ein Ordner main_info gäbe. Sehr unwahrscheinlich, aber nicht auszuschließen. Also soll unser Skript eine Warnung ausgeben, wenn der Ordner bereits existiert und anschließend mit einem Fehlerindikator sich beenden.

Lesen Sie mit:

```
$ echo $?
```

den Status der letzten Programmbeendung aus. Unter Linux beenden Programme standardmäßig mit 0. Jede andere Zahl ist ein Fehlercode. Damit sich unser Skript mit einem Fehler beendet, passen wir den Else-Zweig der If-Verzeigung entsprechend an:

```
else
        echo "Der Ordner existiert bereits." >&2
        echo "Beende ohne Änderung" >&2
        exit 1
fi
```

Speichern und führen Sie das Skript erneut aus, um das fehlerhafte Beenden zu überprüfen. Löschen Sie anschließend den Ordner main_info mitsamt dem Inhalt:

```
$ rm -r main_info
```

Führen Sie das Skript erneut aus. Der Ordner wird erstellt und das Skript beendet sich korrekt. Mit der folgenden Funktion können wir das Skript durch weitere Skripte ausführen lassen, falls beispielsweise eine Festplatte an unser Forensik-System angeschlossen wird.

Die Ausgabe unter Linux unterteilt sich in verschiedene Teilausgaben. Die Standardausgabe ist gedacht für eine normale Ausgabe an den Nutzer. Die Errorausgabe ist, wie der Name beschreibt, für Error gedacht. Im Moment nutzt unser Skript die Standardausgabe für die Error. Ändern wir dies nun:

```
echo „Der Ordner main_info existiert bereits!“ >&2
echo „Beende ohne Änderungen!“ >&2
```

Das Kürzel **&1** ist die Standardausgabe, **&2** die Errorausgabe. Wenn wir nun das Skript mit:

```
$ ./basic_info.sh > /dev/null
```

aufrufen, werden Standardausgaben ignoriert, jedoch Errorausgaben weiterhin ausgegeben. Umgekehrt hat:

```
$ ./basic_info.sh 2> /dev/null
```

den gegenteiligen Effekt. Errorausgaben werden ignoriert und Standardausgaben ausgegeben.
Sie merken, vermutlich schon, dass die Skripte schnell wachsen und man leicht den Überblick verliert. Außerdem kann ein kreativ ausgetüftelter Befehl nach wenigen Tagen nur noch kryptisch erscheinen. Kommentieren wir also unser Skript. Text nach einem # wird bei BASH ignoriert. Gute Praxis ist es nach dem Shebang (#!/bin/bash), dass in der ersten Zeile stehen muss, eine Kurzbeschreibung für das Skript zu geben. Dazu gehört üblicherweise der Autor. Beispielsweise:

```
#!/bin/bash
# Shebang/Erklärung, dass das Skript durch BASH interpretiert wird
```

Shell-Skript verstehen

Sie haben bei der Untersuchung eines beschlagnahmten Datenträgers das folgende Shell-Skript gefunden. Die Ausführung fremden Programmcodes kann zur Infektion Ihres Systems führen. Daher beschließen Sie, das Skript zu untersuchen, indem Sie dessen Funktionsweise anhand des Quellcodes analysieren. Wozu dient das folgende Skript? Gibt es ein Hinweis auf betriebenes illegales Glücksspiel?

```
#!/bin/bash
zahl1=$((RANDOM % 20 +1))
count=0
while [ $count -lt 3 ];
do
        let count=$count+1;
        echo -e "\e[1;31m$count. Versuch""\e[0m"
        read -p "Welche Zahl wird gesucht? : " zahl2
        if [ "$zahl1" -eq "$zahl2" ]
        then
                echo -e "\e[1;31mDein Tipp ist richtig!""\e[0m"
                break
        elif [ "$zahl1" -lt "$zahl2" ]
        then
                echo -e "\e[1;31mDein Tipp ($zahl2) ist groesser als die gesuchte Zahl"
                echo -e "\e[1;31mVersuch es nochmal : ""\e[0m"
        elif [ "$zahl1" -gt "$zahl2" ]
        then
                echo -e "\e[1;31mDein Tipp ($zahl2) ist kleiner als die gesuchte Zahl"
        echo -e "\e[1;31mVersuch es nochmal : ""\e[0m"
        fi
done
if [ "$zahl1" -ne "$zahl2" ]
then
        echo -e "\e[1;31mDein Tipp ist falsch! Die gesuchte Zahl war ($zahl1)" !"\e[0m"
fi
exit 0
```

Shell-Skript verstehen_Lösung

Es handelt sich um ein Skript, das eine Zufallszahl zwischen 1 und (inklusive) 20 generiert. Der Benutzer hat drei Versuche, um die richtige Zahl zu erraten. Das Skript gibt Hinweise aus, ob die zu erratende Zahl höher oder niedriger liegt.

#!/bin/bash	Skript wird durch bash interpretiert
zahl1=$((RANDOM % 20 +1))	Zufällige Zahl zwischen 1 und 20 als Zahl1 fest
count=0	Counter auf 0 setzen(Spielanfang)
while [$count -lt 3];	Solange Versuchscounter weniger als 3 dann:
do	
let count=$count+1;	Counter+1 weil erster Versuch gefragt
echo -e "\e[1;31m$count. Versuch""\e[0m"	Frage ausgeben,
read -p "Welche Zahl wird gesucht? : " zahl2	Antwort einlesen
if ["$zahl1" -eq "$zahl2"]	Wenn Antwort der gesuchten Zahl entspricht,
then	dann
echo -e "\e[1;31mDein Tipp ist richtig!""\e[0m"	Erfolgsnachricht ausgeben
break	und Schleife abbrechen
elif ["$zahl1" -lt "$zahl2"]	Wenn eingegebene Zahl größer als gesuchte
then	Dann
echo -e "\e[1;31mDein Tipp ($zahl2) ist groesser als die gesuchte Zahl"	darüber benachrichtigen
echo -e "\e[1;31mVersuch es nochmal : ""\e[0m"	Und neuen Versuch starten
elif ["$zahl1" -gt "$zahl2"]	Wenn eingegebene Zahl kleiner als gesuchte
then	Dann
echo -e "\e[1;31mDein Tipp ($zahl2) ist kleiner als die gesuchte Zahl"	darüber benachrichtigen
echo -e "\e[1;31mVersuch es nochmal : ""\e[0m"	Und neuen Versuch starten
fi	Ende If Verzweigung
done	Ende While-Schleife
if ["$zahl1" -ne "$zahl2"]	Wenn eingegebene und gesuchte Zahl ungleich
then	Dann Anzeige, dass der Tipp falsch ist
echo -e "\e[1;31mDein Tipp ist falsch! Die gesuchte Zahl war ($zahl1)" !"\e[0m"	Gesuchte Zahl offenbaren
fi	
exit 0	Skript beenden mit Errorcode 0

Zusammensetzung für die Farbcodes siehe Quelle:
https://phlow.de/magazin/terminal/echo/

Shell-Skript lottozahl

In der letzten Aufgabe sollen Sie selbstständig ein ausführbares Skript ***lottozahl*** entwerfen, welches 6 zufällige Zahlen (zwischen 1 und 49) festlegt und anschließend auf der Kommandozeile anzeigt. Beginnen Sie zunächst mit dieser Aufgabe.

Tipp: *Beginnen Sie zunächst zu überlegen, wie eine Zufallszahl im Bereich von 1 bis 49 gebildet werden kann. Anschließend gilt es zu überlegen, wie wir statt einer, sechs Zahlen ausgeben können. Denken Sie auch an die nötigen Rechte und Angaben, die das Skript besitzen muss, um ausgeführt werden zu können.*

Sorgen Sie mit einer Ergänzung nun dafür, dass die Zahlen sortiert ausgegeben werden!

Fügen Sie als letzte Ergänzung folgende Eigenschaft hinzu: Wenn das Skript mit dem Parameter **-s** aufgerufen wird, soll eine zusätzliche Superzahl im Zahlenbereich von 9 bis 39 ausgegeben werden.

Shell-Skript lottozahl_Lösung

```
# Autoren: Leander Hoßfeld, Tim Wetterau
# Version: 27.03.2023
# Dieses kleine Skript erstellt 6 zufällige Zahlen zwischen 1 und 49 und bei Angabe der Flag "-s" eine Superzahl im Bereich von 9 bis 39.

#!/bin/bash                                                    # Shebang
for i in {1..6}                                                # für i im Bereich von 1 bis 6…
do                                                             # mache…
      echo $(( ( RANDOM % 49 ) + 1 ))                          # generiere und gib eine zufällige Zahl.
zwischen 1 und 49 aus
done | sort -n                                                 # Ende der Schleife. Anschließend numerisch sortieren.


if [[ $# -ge 1 && $1 == “-s“ ]]                                # Wenn Anzahl der angegeben Argumente größer/gleich 1 und das erste “-s“ ist…

then                                                           # dann…
      echo Superzahl: $(( ( RANDOM % 31 ) + 9 ))               # generiere und gib eine zufällige Zahl zwischen 9 und 39 aus.

elif [[ $# -ge 1 && $1 != “-s“ ]]                              # Andernfalls, wenn Anzahl der angegeben Argumente größer/gleich1 und das erste nicht “-s“ ist…


then                                                           # dann…
      echo Falsche Flag gesetzt. Gültige Flags: -s             # gib den Fehlertext aus.
fi                                                             # Ende der if-Schleife.

schwarz – Teil 1
rot – Teil 2
grün – Teil 3
```