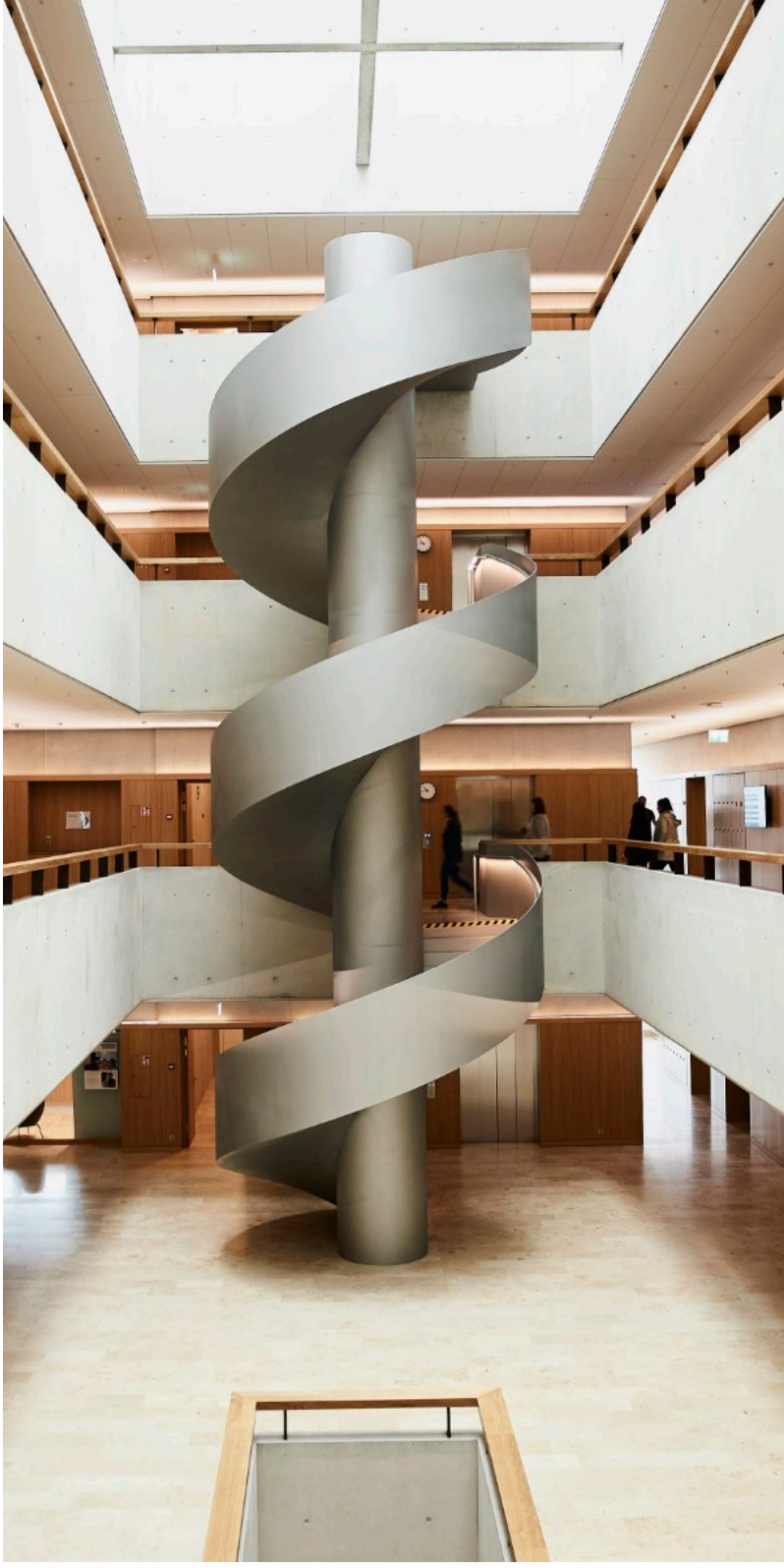
Hochschule Luzern
Modul I.BA_OSSEC



Laborübung

Host Intrusion Detection System

HSLU Hochschule Luzern

Dokument Version: #eb3676d
Template Version: #703559b
Letzte Änderung: 2024-01-29
S. Miescher, J. Zavrlan, M. Useini

I. Allgemeine Informationen

Name:

Gruppe:

Bemerkungen:

Liste der Verfasser

S. Miescher	Erster Entwurf der Laborübung Korrektur & Überprüfung der Laborübung Finalisierung & Korrektur Änderungen & Optimierung Feinschliff & Korrekturen
J. Zavrlan	Ausweitung des Testing-Prozesses
M. Useini	Prüflesen, Testen

Copyright Informationen

II. Inhaltsverzeichnis

III. Vorwort

Feedback

Mit Ihrer Mithilfe kann die Qualität des Versuches laufend den Bedürfnissen angepasst und verbessert werden.

Falls in diesem Versuchsablauf etwas nicht so funktioniert wie es beschrieben ist, melden Sie dies bitte direkt dem Laborpersonal oder erwähnen Sie es in Ihrem Laborbericht oder Protokoll. Behandeln Sie die zur Verfügung gestellten Geräte mit der entsprechenden Umsicht.

Bei Problemen wenden Sie sich bitte ebenfalls an das Laborpersonal.

Legende

In den Versuchen gibt es Passagen, die mit den folgenden Boxen markiert sind. Diese sind wie folgt zu verstehen:

Wichtig

Dringend beachten. Was hier steht, unbedingt merken oder ausführen.

Aufgabe III.1

Beantworten und dokumentieren Sie die Antworten im Laborprotokoll.

Hinweis

Ergänzender Hinweis / Notiz / Hilfestellung.

Information

Weiterführende Informationen. Dies sind Informationen, die nicht zur Ausführung der Versuche benötigt werden, aber bekannt sein sollten.

Story

Hierbei wird die Geschichte vermittelt, die in den Versuch einleitet oder den Zweck des Versuches vorstellt.

Zielsetzung

Lernziele, die nach dem Bearbeiten des Kapitels erfüllt sein sollten.

Erkenntnis

Wichtige Erkenntnisse, die aus dem Versuch mitgenommen werden sollten.

1. Vorbereitung

1.1. Tasks

Zur Vorbereitung für die Laborübung sind folgende Tasks im Vornherein zu erledigen.

- Lesen Sie die Anleitung schon vor dem Labornachmittag durch.

1.2. Benötigte Mittel

Für diesen Versuch wird ein SSH-Zugang zur Laborübungs-VM benötigt. Der SSH-Zugang zur Laborübungs-VM ist nur innerhalb des HSLU-Netzes möglich, von aussen können Sie via VPN zugreifen.

Sämtliche Aktionen lassen sich über die Kommandozeile durchführen, die VM verfügt nicht über eine grafische Oberfläche.

1.3. Abgabe

Keine Abgabe des PDFs nötig! Laden Sie stattdessen den Code auf den bereitgestellten Server hoch.

2. Programmieren in Python

Auch wenn Sie bisher noch nie Python gesehen haben, ist die Sprache dafür geeignet, einfach verständlich zu sein. Durch das Betrachten des bereitgestellten Codes sollten Sie die wichtigsten Spracheigenschaften von Python kennenlernen.

2.1. Python-Versionen

Python gibt es in zwei Hauptversionen, Python 2 und Python 3. Diese unterscheiden sich in der Syntax. Python 2 wird inzwischen nicht mehr unterstützt, es gibt aber noch sehr viel Informationen dazu online. Wenn Sie nach Dokumentation suchen, stellen Sie sicher, dass Sie nur Ergebnisse für Python 3 verwenden.

2.2. Entwicklungsumgebung aufsetzen

Für diese Laborübung müssen Sie Python-Code auf der Labor-VM editieren und ausführen. Auch wenn Sie vor dem Lösen des Labors gerne das Programmieren in Python lernen möchten, können Sie die VM nutzen. Darauf ist ein Python-Interpreter installiert. Sie können über den SSH-Zugang direkt Code im Interpreter schreiben:

```
$ ssh labadmin@\<IP_Adresse_Ihrer_Maschine\>
labadmin@86.119.45.108's password:
 _  _ ___  ____ _ _ ____ ____ _ _ _ _ _   _____
| | | |/ _ \ / ___| | | / ___| / ___| | | | | | | | | ____|
| |_| | | | | | | |_| \___ \| | | |_| | | | | | | | _|
| _ | |_| | |___| _ |___) | |___| _ | |_| | |___| |___
|_| |_|\___/_\____|_|_|_|____/ \____|_| |_|\___/|_____|_____|
| | | | | |__ / ____| _ \| \ | |
| | | | | | / /| _| | |_) | \| |
| |__| |_| |/ /_| |___| _ <| |\ |
|_____\___//____|_____|_| \_\_| \_|

Last login: Thu Jan 01 13:37:00 1970 from 127.0.0.1
labadmin@86.119.45.108:~$ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mylist = ['Hello World', 1234, 'a']
>>> for i in mylist:
...  print(i)
...
Hello World
1234
a
>>>
```

Listing 1: Python3-Interpreter über SSH

Das Schreiben von Code in einem Konsolen-Texteditor über SSH ist zwar möglich, aber umständlich. Es wird empfohlen, Änderungen am Code lokal auf Ihrer Maschine zu machen und die geänderte Datei dann auf dem Server auszuprobieren.

2.2.1. SCP

Zum Kopieren von Dateien zwischen Ihrer lokalen Maschine und dem Server können Sie SCP verwenden. SCP kopiert Dateien zwischen Hosts über SSH. Falls Sie lokal eine Unix-ähnliche Umgebung haben, verwenden Sie das scp-Kommando wie folgt:

```
scp labadmin@\<IP_Adresse_Ihrer_Maschine\>:~/hids.py .
scp ./hids.py labadmin@\<IP_Adresse_Ihrer_Maschine\>:~/
```

Der erste Befehl kopiert die Datei vom Server ins lokale Verzeichnis. Der zweite Befehl kopiert die Datei aus dem lokalen Verzeichnis wieder auf den Server.

Wichtig

Verwenden Sie den Ihnen zugewiesenen Server mit der korrekten Nummer anstelle von *-99* in diesem Beispiel.

Unter Windows können Sie das Programm WinSCP verwenden, dieses kommt mit einer grafischen Oberfläche. Falls Sie MobaXterm als SSH-Client verwenden: Das hat schon ein File Explorer-Feature eingebaut, was im Hintergrund auch scp verwendet. Betriebssystemunabhängig ist auch FileZilla eine Alternative, dabei müssen Sie das SFTP-Protokoll verwenden. In allen Fällen verwenden Sie die Zugangsdaten wie für SSH.

2.2.2. Lokaler Editor

Es steht Ihnen frei, Ihren eigenen Lieblings-Editor zum Bearbeiten des Codes zu verwenden. Unter Windows ist Notepad++ ein schlanker Editor, aber auch IDEs wie Visual Studio Code sind hilfreich. Für GNU/Linux können Sie den mitgelieferten Editor Ihrer Desktopumgebung verwenden, z.B. Kate unter KDE; oder – falls Ihnen schon bekannt – einen Kommandozeilen-Editor wie Emacs oder VIM. Eine gute Option ist auch PyCharm Community Edition.

Verwenden Sie nicht zu viel Zeit auf das Einrichten einer Programmierumgebung! Schlussendlich müssen Sie eine einzige Datei bearbeiten, und ein Texteditor reicht dazu. Bloss notepad.exe ist nicht empfehlenswert, da dies nicht mit Unix-Zeilenenden klarkommt.

Wichtig

Unter Windows ist bei jedem Editor oder IDE wichtig, die **Zeilenenden** im **unix-Format** (\n statt \r\n) zu lassen; oder diese ins unix-Format zu konvertieren, bevor das Ergebnis auf den Server hochgeladen wird. In Notepad++ beispielsweise über *Edit – EOL Conversion – Unix (LF)*

2.3. Python lernen

Die Programmiersprache Python ist bekannt dafür, vergleichsweise einsteigerfreundlich zu sein, und lässt sich zumindest auf einem Basis-Level in kurzer Zeit lernen. Insbesondere wenn Sie schon in einer anderen Sprache programmieren können, sollte das Verstehen und Schreiben von Python-Code problemlos möglich sein.

2.4. Spracheigenschaften

Die folgenden Spracheigenschaften sind wichtig zu kennen und unterscheiden sich möglicherweise von anderen Programmiersprachen, die Sie bereits kennen:

- Kein Semikolon am Zeilenende, das Zeilenende allein separiert zwei Anweisungen
- Code-Blöcke sind durch Einrückung abgetrennt, nicht durch geschweifte Klammern
- Variablen brauchen keine Deklaration oder Datentyp, einfaches Zuweisen erstellt die Variable: a=1
- Strings mit einfachen oder doppelten Anführungszeichen: 'String' oder "String"
- Kommentare mit Raute bis zum Zeilenende: # comment
- Einfache Ausgabe mit print():

```
print("text auf kommandozeile ausgeben")
```

- Formatstrings mit *f""* um den String und *{}* um die Variable innerhalb des Strings:

```
print(f"Variable var1 hat den Wert {var1}")
```

- Logikoperationen mit and und or, Vergleich mit <, <=, >=, >, == und !=, Negation mit not
- If-Statements, Schleifen und Funktionen mit Doppelpunkt am Ende:

```
if liste[1] > 5:
    pass  # do nothing

def myfunction(a, b):
    sum = a+b
    print(f'summe={sum}')
```

- Generisches «contains» mit *in*: *'e' in 'test'*, *2 in liste*, etc.

```
if element in liste:
    print(f'liste enthält {element}')

for el in liste:
    print(f'aktuelles element={el}')
```

- Klassen und OOP-Funktionalität sind für diese Laborübung nicht nötig.

2.4.1. Listen und Dictionaries

Python nennt ein einfaches Array «list» (Liste), die Anzahl Elemente einer Liste ist dynamisch und nicht fix wie in Arrays bei einigen anderen Sprachen. Ein assoziatives Array (Key-Value-Paare) wird «dictionary» genannt. Die Syntax zum Erstellen und Element-Zugriff ist vergleichbar mit JSON.

- Listen mit eckigen Klammern: *my_list = ['a', 3, 'some value']*
- Dictionaries mit geschweiften Klammern: *my_dict = {'key':1, 'other key':'xyz'}*
- Element-Zugriff mit eckigen Klammern: *my_dict['c'] = 'abc' und x = my_list[0]*
- Index beginnt bei 0

2.4.2. Operationen auf Dictionaries

- Test auf Existenz eines Keys: *key in my_dict*
- Test auf Existenz eines Werts: *val in my_dict.values()*
- Hinzufügen oder Überschreiben eines Key-Value-Paars: *my_dict[key]=newvalue*
- Entfernen eines Key-Value-Paars: *Siehe Hinweis in 2.4.4.*

1. Entfernen, wenn der Key garantiert existiert: *del my_dict[key]*
2. Entfernen falls Key existiert, sonst nicht: *my_dict.pop(key, None)*

- Verschachtelte Dictionaries mit mehreren [], z.B. *my_dict[path]['type'] = 'f'*
- Vergleich zweier Dictionaries (gleiche Keys, gleicher Value für jeden Key) mit ==, z.B. *dict_1 == dict_2*

2.4.3. Iteration über Listen und Dictionaries

Iteration über Listen und Dictionaries wird normalerweise mit einem for-in-Loop gemacht. Diese verhält sich unterschiedlich zwischen Listen und Dictionaries. Bei Listen wird als Iterator jeweils das Element aus der Liste verwendet (also Value), bei Dictionaries jeweils der Key (und nicht Value).

Beispiel mit Liste:

```
# Iterate over a List
my_list = ['x','y','z']
for i in my_list:
  print(i)
```

Ausgabe:

```
x
y
z
```

Beispiel mit Dictionary:

```
# Iterate over a Dictionary
my_dict = {'a': 'value1', 'b': 'value2', 'c': 'value3'}
for i in my_dict:
  print(i)
```

Ausgabe:

```
a
b
c
```

Zugriff auf die Values im Dictionary:

```
# Iterate over a Dictionary
my_dict = {'a': 'value1', 'b': 'value2', 'c': 'value3'}
for i in my_dict:
  print(my_dict[i])
```

Ausgabe:

```
value1
value2
value3
```

2.4.4. Element entfernen während Iteration

Entfernen von Elementen aus einer Liste oder einem Dictionary während einer Iteration kann problematisch sein:

```
my_dict = {'a': 'value1', 'b': 'value2', 'c': 'value3'}
for i in my_dict:
  if i == 'b':
    del my_dict[i]
```

Dies gibt einen Fehler («RuntimeError»), weil der for-Loop nicht über etwas iterieren kann, das sich während der Iteration ändert. Verwenden Sie dazu dict.copy():

```
my_dict = {'a': 'value1', 'b': 'value2', 'c': 'value3'}
for i in my_dict.copy():
  if i == 'b':
    del my_dict[i]
```

2.5. Reservierte Namen

Python hat viele eingebaute Funktionen und diverse Keywords, die nicht als Namen für Variablen oder Funktionen verwendet werden sollten. In dieser Laborübung sind besonders die reservierten Namen **file**, **dir** und **hash** erwähnenswert, die Sie vermeiden sollten.

2.6. Nützliche Funktionen für dieses Labor

- **os.path.isfile()** – Überprüft, ob ein Pfad eine existierende Datei ist
- **os.path.isdir()** – Überprüft, ob ein Pfad ein existierendes Verzeichnis ist
- **os.stat()** – Gibt ein Stat-Objekt zurück mit Metainformationen zu einem Pfad
- **pass** – No-Op, d.h. macht nichts. Platzhalter für leere Codeblöcke.

2.7. File modes mit S_IMODE und Nummern in Oktalschreibweise

Als mode wird unter Linux eine Zahl bezeichnet, die Dateiattribute enthält. Darin stehen unter anderem Permission-Bits für die Zugriffsrechte (read,write,execute) für user, group und andere Nutzer, sowie der Dateityp und weitere Metadaten. Das **stat**-Objekt, welches von **os.stat()** zurückgegeben wird, liefert die komplette mode-Nummer als Datentyp **integer** (Ganzzahl). Uns interessieren aber nur die Permission-Bits. Ausserdem sollen diese in Oktal-Darstellung gespeichert werden – Dateizugriffsrechte unter Linux sind generell immer in Oktalschreibweise angegeben. Es müssen also zuerst die Permission-Bits extrahiert werden und danach die Zahl zu einem Oktal-String konvertiert. Dazu stehen in Python diese zwei Funktionen zur Verfügung:

- **stat.S_IMODE()** – Extrahiert Permission-Bits (unterste 9 bits) aus einer Zahl
- **oct()** – Konvertiert eine Zahl zu einem String in Oktalschreibweise

3. Host Intrusion Detection System

In dieser Übung werden Sie ein einfaches Host Intrusion Detection System (HIDS) programmieren. Die Anleitung führt Sie diesmal nicht schrittweise durch die Übung, sondern zeigt Ihnen nur das Ziel und einige Hinweise. Den Rest sollten Sie sich selbständig erarbeiten.

Das Programm ist für diese Laborübung in Python geschrieben, was generell einfacher verständlich und weniger fehleranfällig ist als C-Code.

3.1. Aufgabe

Ziel ist es, ein Programm zu schreiben, welches Dateien und Datei-Attribute findet, speichert, vergleicht und Änderungen bemerkt. Das Intrusion Detection System findet also Veränderungen im Dateisystem. Auf einem echten Produktivsystem würde regelmässig ein Scan auf Änderungen durchgeführt.

Der Workflow bei der Benutzung des Programms ist wie folgt:

1. Erstellen einer Datenbank für einen Verzeichnisbaum mit einer Liste von Dateien und Verzeichnissen und deren Attribute;
2. (Optional) berechnen von Prüfsummen für Dateien;
3. Produktiv-Einsatz des Systems – simuliert durch eigene Änderungen (im Idealfall ändert sich nichts, eine Änderung würde auf eine Intrusion hindeuten);
4. Überprüfen des Verzeichnisbaums auf Änderungen in Dateien und Attributen, Erkennen von hinzugefügten, veränderten oder entfernten Dateien und Verzeichnissen.

Zudem soll es Veränderungen erlauben, wenn dies gewünscht ist:

1. Gewollte Änderungen an Dateien vornehmen (z.B. Einspielen von Updates) und diese Änderungen zur Datenbank hinzufügen;
2. Weiter bei 3.

3.2. Template

Sie finden in der Datei hids.py bereits ein Skelett für die Implementierung eines HIDS. Sie können das Programm mit der folgenden Syntax starten:

```
1 python3 hids.py -d DATENBANK -p DATEI_ODER_VERZEICHNIS AKTION
```

Bereits vollständig implementiert ist die Aktion «count», mit der lediglich die Anzahl Dateien und Verzeichnisse gezählt werden. Für diese Aktion ist keine Datenbank nötig. Testen Sie diese Aktion:

```
1 python3 hids.py -p ~ count
```

Sie sollten als Rückgabe die Anzahl Dateien und Verzeichnisse im Home-Verzeichnis des aktuellen Benutzers erhalten.

3.3. Datenbankformat

Als «Datenbank» wird eine einfache JSON-Datei verwendet – also keine echte Datenbank. Diese lässt sich dafür im Zweifelsfall manuell bearbeiten und trivial löschen oder neu erstellen. Der Inhalt soll folgende Struktur haben:

```
{
   '/path/to/file/file1.txt' :
   {
      'type': 'f',
      'uid': 1234,
      'gid': 1234,
      'mode': '0o644',
      'size': 999,
      'hash': '0123456789ABCDEF'
   },
   '/path/to/dir1' :
   {
      'type': 'd',
      'uid': 1234,
      'gid': 1234,
      'mode': '0o755'
   },
   ...
}
```

Innerhalb des Codes entspricht ein JSON-Objekt einem Python-Dictionary. Beide werden mit geschweiften Klammern geschrieben und können als assoziatives Array verstanden werden. Die Konvertierung zwischen der JSON-Datei («Datenbank») und dem Dictionary innerhalb des Codes ist bereits implementiert.

3.4. Dateiattributen

Folgende Attribute sollten Sie für jede Datei speichern:

1. Benutzer-ID (uid)
2. Gruppen-ID (gid)
3. Zugriffsrechte (mode)
4. Dateigrösse (size)
5. Prüfsumme (hash), nur falls vom Benutzer gewünscht

Für Verzeichnisse speichern Sie die folgenden Attribute:

1. Benutzer-ID (uid)
2. Gruppen-ID (gid)
3. Zugriffsrechte (mode)

3.5. Aktionen

Die Aktion «count» ist bereits vorhanden. Vier weitere Aktionen sollen von Ihnen implementiert werden.

Jede der Aktionen hat diese Funktionsdeklaration:

```
def action(data, files, directories):
```

In data wird das Dictionary der bereits vorhandenen Datenbank übergeben. Dieses kann auch leer sein.

In files wird eine Liste aller Dateien (als Dateipfad) angegeben, die im Verzeichnis drin sind, das vom Benutzer bei der Ausführung mit dem Parameter -p angegeben wird. In directories ist ensprechend eine Liste aller Verzeichnisse.

3.5.1. add

Neue Dateien und Verzeichnisse zur Datenbank hinzufügen.

Falls eine Datei oder ein Verzeichnis schon in der Datenbank existiert, ist dies ein Fehler – zum Ändern von Einträgen soll die «update»-Aktion verwendet werden. Falls zu einer existierenden Datenbank eine zusätzliche Datei hinzugefügt werden soll, kann mit -p der exakte Dateipfad spezifiziert werden.

Ein Grossteil dieser Funktionalität ist bereits implementiert. Zwei Hilfsfunktionen werden dafür verwendet: dirdata() und filedata(). Diese sind noch nicht komplett – vervollständigen Sie diese zwei Funktionen, um die Aktion «add» fertig zu implementieren.

Für die weiteren Aktionen können Sie sich grob an der Struktur von «add» orientieren.

3.5.2. hash

Prüfsumme (englisch «hash») für Dateien (aber nicht Verzeichnisse) in der Datenbank berechnen und darin speichern.

Dies könnte auch als Teil der add-Funktion implementiert werden, Prüfsummen berechnen dauert aber länger und wird deshalb möglicherweise nicht für alle Dateien gemacht.

Für die Berechnung der Prüfsumme anhand eines Dateipfades verwenden Sie die bereits implementierte Funktion sha256file().

Verzeichnisse haben keine Prüfsumme und werden deshalb in dieser Funktion ignoriert.

Die Funktions-Logik soll folgendermassen sein:

1. Gegeben sei eine Datenbank und eine Liste von Dateien;
2. Für jede Datei (nicht für Verzeichnisse):
    1. Falls die Datei nicht in der Datenbank existiert, ist dies ein Fehler => Abbruch;
    2. Falls der Datenbank-Eintrag der Datei schon einen Hash enthält (Key ‘hash’ existiert), ist dies ein Fehler => Abbruch;
    3. Berechne Prüfsumme der Datei und füge diese zur Datenbank hinzu, indem für die aktuelle Datei ein neuer Key ‘hash’ hinzugefügt wird mit der Prüfsumme als Value.
3. Ausgabe der Anzahl Dateien, für die erfolgreich eine Prüfsumme berechnet wurde;
4. Die Funktion soll «False» zurückgeben bei einem Abbruch, sonst «True».

3.5.3. update

Wie add, aber für bereits existierende Dateien werden alle vorhandenen Attribute aktualisiert und überschrieben.

Dies ist eine separate Funktion, damit nicht aus Versehen Werte überschrieben werden.

Die Funktions-Logik soll folgendermassen sein:

1. Gegeben sei eine Datenbank, eine Liste von Dateien und eine Liste von Verzeichnissen;
2. Für jeden Eintrag in der Datenbank:
    1. Falls der Datenbank-Eintrag nicht in der Liste von Dateien oder Verzeichnissen vorkommt, entferne diesen Eintrag aus der Datenbank.
3. Für jede Datei und jedes Verzeichnis:
    1. Falls Datei oder Verzeichnis nicht in der Datenbank existiert, analog zu add;
    2. Falls Datei oder Verzeichnis in der Datenbank existiert:
        - Alle Werte neu einlesen und vorhandene Werte damit überschreiben;
        - Falls es sich um eine Datei handelt und diese einen Hash besitzt, Hash komplett entfernen. Falls der Benutzer wieder Hashes haben möchte, muss er nochmals die «hash»-Aktion verwenden.
4. Ausgabe der Anzahl Dateien und Verzeichnisse, die neu hinzugefügt wurden;
5. Ausgabe der Anzahl Dateien und Verzeichnisse, deren Eintrag geändert wurde (entfernter Hash ignorieren);
6. Ausgabe der Anzahl Dateien mit entferntem Hash;
7. Ausgabe der Anzahl Dateien und Verzeichnisse, die entfernt wurden;
8. Die Funktion soll immer «True» zurückgeben.

3.5.4. verify / check

Werte in der Datenbank mit dem Dateisystem abgleichen.

Mit dieser Funktion wird nun tatsächlich überprüft, ob sich etwas unerwartet geändert hat, also «Intrusion Detection».

Die Funktions-Logik soll folgendermassen sein:

1. Gegeben sei eine Datenbank, eine Liste von Dateien und eine Liste von Verzeichnissen;
2. Für jeden Eintrag in der Datenbank:
    1. Falls der Datenbank-Eintrag nicht in der Liste von Dateien oder Verzeichnissen vorkommt, gebe eine Warnung «missing» inkl. Pfad aus.
3. Für jede Datei und jedes Verzeichnis:
    1. Falls Datei oder Verzeichnis nicht in der Datenbank existiert: Ausgabe einer Warnung «new file»/«new directory» mit Dateinamen;
    2. Falls Datei oder Verzeichnis in der Datenbank existiert:

- In der Datenbank gegebene Werte (ausser Hash) mit der aktuellen Datei vergleichen;
- Falls ein oder mehrere Werte nicht übereinstimmen zwischen Datenbank und Dateisystem:
  - Ausgabe einer Warnung «mismatch» mit Pfad;
  - Ausgabe des geänderten Attributs, Wert in der Datenbank und neuer Wert.
- Falls der Datenbankeintrag einen Hash hat (nur bei Dateien):
  - Hash der Datei neu berechnen und mit Datenbankeintrag vergleichen;
  - Falls Hash unterschiedlich:
    * Ausgabe einer Warnung «hash changed» mit Pfad;
    * Ausgabe des alten und des neuen Hashes.

4. Falls es mindestens eine Warnung gegeben hat:

    1. Ausgabe der Anzahl problematischen Dateien und Verzeichnisse je Warnungs-Typ.

5. Die Funktion soll «True» zurückgeben, falls keine Probleme gefunden wurden, sonst «False».

Vergessen Sie nicht zu überprüfen, ob sich der Typ (Datei oder Verzeichnis) geändert hat!

4. Testen

Sobald Sie fertig mit Coden sind, sollten Sie folgende Aktionen mit zu erwartenden Resultaten durchführen können:

1. Datenbank in **/tmp/testdb** anlegen und alle Dateien im Heimverzeichnis (**~**) hinzufügen;
2. Zusätzliche Datei mit Inhalt «abc» im Heimverzeichnis anlegen;
3. Nochmals add auf Heimverzeichnis, sollte Fehler ausgeben, weil Dateien schon vorhanden sind;
4. Check auf Heimverzeichnis, sollte Fehler ausgeben, weil neue Datei existiert;
5. Add für nur die eine neue Datei;
6. Check auf Heimverzeichnis, sollte alles korrekt sein;
7. Dateiinhalt zu «abcd» ändern;
8. Nochmals Check, sollte Fehler ausgeben (Dateigrösse ändert sich);
9. Update für Heimverzeichnis, sollte eine neue Datei erkennen;
10. Nochmals Check, sollte alles korrekt sein;
11. Berechtigungen für die Datei ändern (z. B. chmod 777 file.txt);
12. Nochmals Check, sollte Fehler ausgeben (Dateiberechtigungen ändern sich);
13. Update für Heimverzeichnis, sollte eine neue Datei erkennen;
14. Nochmals Check, sollte alles korrekt sein;
15. Dateiinhalt zu «1234» ändern;
16. Nochmals Check, sollte immer noch korrekt sein (gleiche Grösse, deshalb brauchen wir Hashes);
17. Hashes hinzufügen;
18. Check, sollte alles korrekt sein;
19. Dateiinhalt zu «abcd» zuruckändern;
20. Check, sollte Fehler anzeigen;
21. Update auf das Heimverzeichnis, keine Datei sollte sich ändern, aber Hashes entfernt;
22. Wieder Hashes hinzufügen;
23. Check sollte korrekt sein;
24. Datei löschen;
25. Check sollte das Fehlen der Datei anzeigen.

Wenn das alles einwandfrei funktioniert, können Sie den Code abgeben.

5. Was man aus der heutigen Lektion mitnehmen sollte

Sie haben ein einfaches Host Intrusion Detection System implementiert. Sie kennen dessen Funktionsweise und Limits. Im Produktiveinsatz sollte ein gängiges Tool eingesetzt werden, das stabil läuft und verbreitet ist. Idealerweise verwenden Sie ein Open Source-Tool, für das Sie im Zweifelsfall den Quellcode verifizieren und somit dem Tool vertrauen können.

Damit das Tool nicht in einer Umgebung läuft, in der Sie die Möglichkeit einer Intrusion haben, könnte man beispielsweise Scans auf einen Container von ausserhalb des Containers laufen lassen, oder Scans einer VM von ausserhalb der VM, indem das VM-Dateisystem im Nur-Lese-Modus gemountet wird.

Generell sind IDS anfällig für false negatives. Deshalb gilt immer: Ein IDS kann Probleme finden, aber es kann diese nicht ausschliessen. Schlussendlich ist ein IDS nur eine Sicherheitsmassnahme, die gegen bestimmte Arten von Angriffen hilft, aber nicht gegen alle.

Notizen