Evaluare Curs - Programare Funcțională
Anul II
Grupa: 202
Nume și prenume student: Manolache Mihnea Octavian

PROIECT PREM_14
WebChecker - Verificarea Statusului Aplicatiilor Web

#1: Prezentare Generală

Am ales implementarea proiectului „WebChecker” în limbajul Python, pentru care am folosit următoarele librării:

- **Requests** - Realizare solicitări HTTP
- **Socket** - Interfață networking low level
- **Argparse** - Utilizare argumente în linia de comandă
- **Termcolor** - Personalizare a textului afișat în terminal
- **Os** - Execuție diverse comenzi ale sistemului de operare

Când discutăm despre analiza statusului unei aplicații web, primul aspect pe care dorim să îl identificăm este dacă respectiva aplicație web este disponibilă pe internet. Totuși, o analiză completă a unei aplicații web ar trebui să ne furnizeze cât mai multe date referitoare la aplicația respectivă. În acest sens, programul WebChecker returnează utilizatorului următoarele informații:

- Verifică dacă URL-ul introdus este valid (dacă aplicația este online)
- Încearcă să identifice dacă aplicația rulează unul din următoarele 4 CMS-uri populare:
    - WordPress
    - Joomla
    - Magento
    - Drupal
- Furnizează statusul HTTP al aplicației web
- Furnizează informații despre tipul serverului pe care este găzduită aplicația
- Furnizează IP-ul serverului pe care este găzduită aplicația
- Furnizează antetul HTTP complet
- Încearcă să identifice alte directoare populare, folosind metoda brute-force

#2: Mediul Virtual

Pentru administrarea librăriilor utilizate în cod, am creat în primul rând un mediu virtual izolat folosind comanda:

```
python3 -m venv env
```

Astfel, pentru rularea programului în terminal, este necesar să navigăm către directorul programului și activarea mediului virtual. În acest sens, vom deschide o nouă fereastră terminal și vom utiliza comanda:

```
cd .../path_to_dubdirectory/WebChecker && source env/bin/activate
```

** Pentru rularea aplicației în PowerShell (Windows) comenzile pot fi diferite*

#3: Analiza Codului Sursă

3.1. Importarea librăriilor în program

Pentru a eficientiza implementarea programului, vom importa în primul rând librăriile necesare pentru execuția acestuia:

```
import requests
import socket
import argparse
from termcolor import cprint
import os
from pprint import pprint
```

3.2. Adăugarea arumentelor

După cum vom observa pe parcurs, aplicația permite introducerea unei aplicații țintă după rulare, sau poate utiliza argumente în comanda terminal.

```
# ADAUGARE ARGUMENTE
parser = argparse.ArgumentParser()
parser.add_argument("-u", "--url", help="Specify the URL ypu want to Check")
parser.add_argument("-b", "--brute", help="Attempt to bruteforce interesting directories")
args = parser.parse_args()
```

- Argumentul „-u” este echivalent cu „--url” și permite specificarea URL-ului aplicației țintă.
- Argumentul „-b” este echivalent cu „--brute” și se utilizează pentru a specifica dicționarul utilizat pentru tehnica brute-force asupra directoarelor aplicației

3.3. Variabile globale

Definim variabilele pe care le vom utiliza în program:

```
# OBTINE LATIMEA TERMINALULUI
```

```
term_size = os.get_terminal_size()


# VARIABILE GLOBALE
GLOBALS = AttrDict({
   "CMS_ERROR": True,
   "WP": False,
   "JOOM": False,
   "MAG": False,
   "DRUP": False,
   "CONFIDENCE":0,
   "MESSAGES": []
})
USER_AGENT = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0;
WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36',}
```

- **GLOBALS** - Obiect în care stocăm mai multe proprietăți:
  - **CMS_ERROR** - Variabilă de tip boolean, utilizată în cazul în care aplicația țintă nu rulează niciunul dintre cele 4 CMS-uri populare
  - **WP** - Variabilă de tip boolean, utilizată în cazul în care aplicația țintă rulează pe platforma WordPress
  - **JOOM** - Variabilă de tip boolean, utilizată în cazul în care aplicația țintă rulează pe platforma Joomla
  - **MAG** - Variabilă de tip boolean, utilizată în cazul în care aplicația țintă rulează pe platforma Magenta
  - **DRUP** - Variabilă de tip boolean, utilizată în cazul în care aplicația țintă rulează pe platforma Drupal
  - **CONFIDENCE** - Variabilă de tip int, utilizată pentru calcularea nivelului de încredere al identificării CMS-ului
- **USER_AGENT** - Variabilă de tip dictionary, utilizată pentru a specifica un tip de agent către aplicația web verificată
- **BANNER** - Stochează banner-ul afișat la inițializarea programului
- **term_size** - Obține dimensiunea ferestrei terminal, utilizată pentru a printa mai apoi o linie verticală de la un cap la altul al ferestrei terminal, ajustabilă în funcție de dimensiunea acestuia.

3.4. Definirea Funcțiilor

3.4.1. Inițializarea programului

La inițializarea programului, folosim comanda „clear” pentru a afișa în terminal doar programul WebChcker și nu comenzile precedente, după care printăm banner-ul și linia verticală ajustabilă:

```
def clear_screen():
    if os.name in ('nt', 'dos'):
        command='cls'
    else:
        command='clear'
    os.system(command)
```

3.4.2. Conversia URL-ului introdus de utilizator

Funcția de conversie a URL-ului introdus de utilizator preia parametrul „url” și îl formează astfel încât să poată fi utilizat de librăriile „requests” și „socket”.Rezultatul returnat de funcția *convert_url* este un obiect cu două proprietăți pe care le vom utiliza ulterior:

```
def convert_url(url):
     # Conversia inputului in URL utilizabil
    if url.startswith('http'):
        url_request = url
        url_socket = url.lstrip('http')
        url_socket = url_socket.lstrip('s')
        url_socket = url_socket.lstrip('://')
        url_socket = url_socket.rstrip('/')
    else:
        url_request = 'http://' + url
        url_socket = url
    return {
                "url_request": url_request,
                "url_socket": url_socket
            }
```

- Request primește parametrii de tip „http://exemplu.com/” sau „https://exemplu.com/”
- Socket primește arametrii fără atributele „http”

3.4.3. Obținerea IP-ului și a unor date din *headers*

Definim trei funcții care vor returna programului diverse informații din *header*-ul obținut în urma unui request http:

```
def get_headers(url_request):
        r   =   requests.get(url_request,   allow_redirects=True,
headers=USER_AGENT)
    return r.headers


def get_status(url_request):
```

```
        r   =   requests.get(url_request,   allow_redirects=True,
headers=USER_AGENT)
    return r.status_code


def get_server_type(url_request):
        r   =   requests.get(url_request,   allow_redirects=True,
headers=USER_AGENT)
    return r.headers['Server']


def get_ip_address(url_socket):
    ip_address = socket.gethostbyname(url_socket)
    return ip_address
```

3.4.4. Printarea rezultatelor:

La finalul programului, observăm că mai multe acțiuni necesită repetarea aceluiași cod, cu mici modificări. Astfel, definim funcția atribuind parametrul *results*, care va acoperi micile diferențe menționate:

```
def print_results(results):
    for i in results.MESSAGES:
        if "[+]" in i:
            cprint(f"    {i}", 'green')
        elif "[-] in i":
            cprint(f"    {i}", 'red')
```

3.4.5. Funcții care verifica CMS-urile populare

CMS-urile (content management system) sunt platforme care permit administrarea conținutului. Principalele CMS-uri în prezent sunt WordPress, Joomla, Magento și Drupal. Vom încerca să identificăm dacă aplicația țintă rulează unul dintre aceste CMS-uri transmițând solicitări către URL-uri predefinite utilizate de aplicațiile respective, precum:

- https://*exemplu.com*/wp-login.php - Pagina de logare în platforma WordPress
- https://*exemplu.com*/wp-admin/upgrade.php - Pagina de upgrade a platformei WordPress
- https://*exemplu.com*/wp-json/wp/v2/ - Pagina API a aplicațiilor WordPress
- https://*exemplu.com*/administrator/ - string-ul „mod-login-username” în textul acestei pagini sugerează de regulă faptul că aplicația rulează pe platforma Joomla
- https://*exemplu.com*/index.php/ - string-urile „/mage/” sau „magento” în textul acestei pagini sugerează de regulă faptul că aplicația rulează pe platforma Magento
- https://*exemplu.com*/readme.txt/ - string-ul „drupal” în textul acestei pagini sugerează de regulă faptul că aplicația rulează pe platforma Drupal

Funcțiile definite pentru descoperirea acestor CMS-uri preiau parametrii: url_request, user_agent, result. Parametrul *results* este echivalent cu variabila *GLOBALS* și este de altfel obiectul principal, returnat de funcție:

```
def check_wp(url_request, user_agent, result):
      wp_url = requests.get(url_request + '/wp-login.php',
allow_redirects=True, headers=user_agent)
      if wp_url.status_code == 200 and "user_login" in
wp_url.text and "404" not in wp_url.text:
      result["CMS_ERROR"] = False
      result["WP"] = True
      result["CONFIDENCE"] += 25
         result["MESSAGES"].append(f"[+] WordPress login page
available at {url_request}/wp-login.php")
   else:
       result["MESSAGES"].append("[-] WordPress login page not
available")


              wp_url     =     requests.get(url_request     +
'/wp-admin/upgrade.php',                allow_redirects=False,
headers=user_agent)
   if wp_url.status_code == 200 and "404" not in wp_url.text:
      result["CMS_ERROR"] = False
      result["WP"] = True
      result["CONFIDENCE"] += 25
         result["MESSAGES"].append(f"[+] WP-Admin/upgrade.php
page available at {url_request}/wp-admin/upgrade.php")
   else:
          result["MESSAGES"].append("[-] WP-Admin/upgrade.php
page doesn't seem to be available")


     wp_url = requests.get(url_request + '/wp-json/wp/v2/',
allow_redirects=False, headers=user_agent)
   if wp_url.status_code == 200 and "404" not in wp_url.text:
      result["CMS_ERROR"] = False
      result["WP"] = True
      result["CONFIDENCE"] += 25
         result["MESSAGES"].append(f"[+] WP API available at
{url_request}/wp-json/wp/v2/")
   else:
      result["MESSAGES"].append("[-] WP API not available")
```

```
        wp_url   =   requests.get(url_request   +   '/robots.txt',
allow_redirects=True, headers=user_agent)
    if wp_url.status_code == 200 and "wp-admin" in wp_url.text:
        result["CMS_ERROR"] = False
        result["WP"] = True
        result["CONFIDENCE"] += 25
            result["MESSAGES"].append(f"[+]  Robots.txt  fount  at
{url_request}/robots.txt containing 'wp_admin'")
    else:
        result["MESSAGES"].append("[-] Robots.txt not found")

    return result

def check_joom(url_request, user_agent, result):
    result["MESSAGES"].clear()
    joom_url = requests.get(url_request + '/administrator/')
     if joom_url.status_code == 200 and "mod-login-username" in
joom_url.text and "404" not in joom_url.text:
        result["CMS_ERROR"] = False
        result["JOOM"] = True
        result["CONFIDENCE"] += 100
         result["MESSAGES"].append(f"[+] {url_request} seems to
be running on Joomla")
    else:
          result["MESSAGES"].append(f"[-] {url_request} doesn't
seem to be running on Joomla")

    return result

def check_mag(url_request, user_agent, result):
    result["MESSAGES"].clear()
        mag_url   =   requests.get(url_request   +   '/index.php',
allow_redirects=False)
     if mag_url.status_code == 200 and '/mage/' in mag_url.text
or 'magento' in mag_url.text:
        result["CMS_ERROR"] = False
        result["MAG"] = True
        result["CONFIDENCE"] += 25
```

```
                result["MESSAGES"].append("[+] Magento strings
detected.")
    else:
            result["MESSAGES"].append("[-] No Magento strings
detected")


    mag_url = requests.get(url_request + '/index.php/admin/',
allow_redirects=False)
    if mag_url.status_code == 200 and 'login' in mag_url.text
and "404" not in mag_url.text:
        result["CMS_ERROR"] = False
        result["MAG"] = True
        result["CONFIDENCE"] += 25
        result["MESSAGES"].append(f"[+] Potential Magento admin
login at {url_request}/index.php/admin/")
    else:
                                result["MESSAGES"].append(f"[-]
{url_request}/index.php/admin/ not available")


    mag_url = requests.get(url_request + '/RELEASE_NOTES.txt')
        if mag_url.status_code == 200 and 'magento' in
mag_url.text:
        result["CMS_ERROR"] = False
        result["MAG"] = True
        result["CONFIDENCE"] += 25
                    result["MESSAGES"].append(f"[+] Magento
Release_Notes.txt                detected                at
{url_request}/RELEASE_NOTES.txt")
    else:
                    result["MESSAGES"].append(f"[-] Magento
Release_Notes.txt not detected")


    mag_url = requests.get(url_request + '/js/mage/cookies.js')
        if mag_url.status_code == 200 and "404" not in
mag_url.text:
        result["CMS_ERROR"] = False
        result["MAG"] = True
        result["CONFIDENCE"] += 25
            result["MESSAGES"].append(f"[+] Magento cookies.js
detected at {url_request}/js/mage/cookies.jst")
    else:
```

```
        result["MESSAGES"].append("[-] Magento cookies.js not
detected")

    return result

def check_drup(url_request, user_agent, result):
    result["MESSAGES"].clear()
    drup_url = requests.get(url_request + '/readme.txt')
        if drup_url.status_code == 200 and 'drupal' in
drup_url.text and '404' not in drup_url.text:
        result["CMS_ERROR"] = False
        result["DRUP"] = True
        result["CONFIDENCE"] += 33
            result["MESSAGES"].append(f"[+] Drupal Readme.txt
detected at {url_request}/readme.txt")
    else:
        result["MESSAGES"].append("[-] Drupal Readme.txt not
detected")

    drup_url = requests.get(url_request)
      if drup_url.status_code == 200 and 'name="Generator"
content="Drupal' in drup_url.text:
        result["CMS_ERROR"] = False
        result["DRUP"] = True
        result["CONFIDENCE"] += 33
                result["MESSAGES"].append("[+] Drupal strings
detected.")
    else:
            result["MESSAGES"].append("[-] No Drupal string
detected")

            drup_url = requests.get(url_request +
'/modules/README.txt')
        if drup_url.status_code == 200 and 'drupal' in
drup_url.text and '404' not in drup_url.text:
        result["CMS_ERROR"] = False
        result["DRUP"] = True
        result["CONFIDENCE"] += 33
        result["MESSAGES"].append(f"[+] Drupal modules detected
at {url_request}/modules/README.txt")
```

```
    else:
            result["MESSAGES"].append("[-] No Drupal modules
detected")

    return result
```

3.5. Inițializarea programului

3.5.1. Verificarea existenței argumentului *url*

Dacă utilizatorul a specificat un URL țintă în comanda terminal, vom utiliza string-ul respectiv pentru a inițializa procedura de verificare a aplicației. În caz contrar, programul va afișa mesajul „Site to check:” solicitând utilizatorului să introducă URL-ul țintă.

```
if args.url is None:
        # Get the input from the user
        url = input('Site to check: ')
        cprint('-'*term_size.columns, 'blue')
else:
    url = args.url
```

3.5.2. Definirea variabilelor necesare programului

Vom defini 4 variabile egale cu rezultatul unora dintre funcțiile definite anterior:

```
url_request = convert_url(url)["url_request"]
url_socket = convert_url(url)["url_socket"]
r = get_headers(url_request)
ip_address = get_ip_address(url_socket)
```

3.5.3. Afișarea unui mesaj inițial în terminal

Vom anunța utilizatorul programului cu privire la faptul că procedura de verificare a aplicației a fost demarată. De asemenea, dacă request-ul inițial a avut succes, printăm mesajul *„Aplicația* pare să fie online”:

```
cprint(f'[*] Checking: {url}...', 'yellow')
cprint(f'[!] {url} seems to be online', 'green')
print('')
```

** Comanda „cprint” face parte din librăria termcolor și permite formatarea textului afișat în consola terminal.*

3.5.4. Verificarile initiale (identificarea CMS-ului)

Folosind functiile definite anterior, vom proba eventualele CMS-uri:

```
    cprint(f'[*] Checking: {url}...', 'yellow')
```

```
  cprint(f'[!] {url} seems to be online', 'green')
   print('')


     cprint("[*] Attempting to identify potential CMS...",
'yellow')
   # Efectuare verificari standard pt WordPress
   cprint("[!] Running WordPress scans...", 'magenta')
     wp_results = AttrDict(check_wp(url_request, USER_AGENT,
GLOBALS))
   print_results(wp_results)


   # Efectuare verificari standard pt Joomla
   cprint("[!] Running Joomla scans...", 'magenta')
   joom_results = AttrDict(check_joom(url_request, USER_AGENT,
GLOBALS))
   print_results(joom_results)


   # Efectuare verificari standard pt Magento
   cprint("[!] Running Magento scans...", 'magenta')
    mag_results = AttrDict(check_mag(url_request, USER_AGENT,
GLOBALS))
   print_results(mag_results)


   # Efectuare verificari standard pt Drupal
   cprint("[!] Running Drupal scans...", 'magenta')
   drup_results = AttrDict(check_drup(url_request, USER_AGENT,
GLOBALS))
   print_results(mag_results)
```

3.5.5. Afișarea rezultatelor în terminal

După efectuarea tuturor verificărilor, vom afișa în terminal rezultatul acestora astfel:

- Statusul aplicației (online/offline)
- Tipul platformei CMS pe care rulează (dacă este cazul)
- Statusul HTTP
- Tipul serverului pe care este găzduită aplicația
- IP-ul serverului pe care este găzduită aplicația
- Antetul HTTP complet

```
   cprint('App overview:', 'magenta')
```

```
    cprint(f'    [+] {url} is available', 'green')
    if GLOBALS.CMS_ERROR:
        cprint(f"    [!] {url} doesn't seem to run any known CMS", 'yellow')
    elif GLOBALS.WP:
        cprint(f"    [+] {url} seems to be running WordPress. [Confidence: {GLOBALS.CONFIDENCE}%]", 'green')
    elif GLOBALS.JOOM:
        cprint(f"    [+] {url} seems to be running Joomla. [Confidence: {GLOBALS.CONFIDENCE}%]", 'green')
    elif GLOBALS.MAG:
        cprint(f"    [+] {url} seems to be running Magento. [Confidence: {GLOBALS.CONFIDENCE}%]", 'green')
    elif GLOBALS.DRUP:
        cprint(f"    [+] {url} seems to be running Drupal. [Confidence: {GLOBALS.CONFIDENCE}%]", 'green')
    cprint(f'    [+] HTTP Status:    {get_status(url_request)}', 'green')
    cprint(f'                [+]    Server    type: {get_server_type(url_request)}', 'green')
    cprint(f'                [+]    Server    IP: {get_ip_address(url_socket)}', 'green')
    print('')
    cprint('-'*term_size.columns, 'blue')


    # ------ AFISARE FULL HTTP HEADERS IN TERMINAL ------ #
    cprint('Detailed HTTP header report:', 'magenta')
    for i in get_headers(url_request): # -> Printare Headers pe linii (prettyprint)
        cprint(f"    [*] {i}: {get_headers(url_request)[i]}", 'cyan')
    print('')
```

3.11. Tratarea excepțiilor

Sintaxa „try” utilizată la început permite inițializarea programului doar dacă URL-ul furnizat este valid. În cazul în care acesta este invalid, vom trata această excepție, afișând utilizatorului mesajul „Eroare”, dar și eroarea furnizată de sistem:

```
except requests.exceptions.RequestException as e:
    cprint(f'[!] Checking: {url}...', 'yellow')
    print('')
```

```
    cprint('ERROR! It seems like the URL you provided is
incorrect or the WebApp is not connected to the Internet',
'red')
   print('')
   cprint('ERROR MESSAGE:', 'yellow', end=' ')
   raise SystemExit(e)
```

3.11. Descoperirea directoarelor interesante

Dacă utilizatorul a folosit argumentul „-b” sau „--brute”, vom aplica tehnica brute force pentru a descoperi eventuale directoare ale aplicației web. Pentru aceasta, odată cu argumentul, utilizatorul trebuie să specifice un dicționar de tip txt, care conține cuvintele pe care dorește să le descopere.

Aplicația va încerca să atașeze fiecare cuvânt URL-ului țintă, iar dacă statusul întors este 200, atunci va informa utilizatorul cu privire la faptul că adresa este validă.

```
if args.brute:
   cprint('-'*term_size.columns, 'blue')
   cprint('Bruteforcing interesting directories:', 'magenta')
   f = open(args.brute, 'r')
   for i in f:
       brut_url = requests.get(url_request + "/" + i.rstrip(),
allow_redirects=True, headers=user_agent)
       if brut_url.status_code == 200:
             cprint(f"    [+] {i.rstrip()} -> {brut_url.url}",
'green')
       else:
             cprint(f"    [-] {i.rstrip()} -> {brut_url.url}",
'red')
```

#4: Rularea Programului

Programul WebChecker rulează direct din terminal, iar execuția se face în Python. Așadar, pentru a putea executa WebChecker.py utilizatorul trebuie să aibă instalată versiunea Python 3. Pentru a verifica dacă Python3 este disponibil în sistem, vom executa următoarea comandă în terminal:

```
~/Desktop/WebChecker » which python3
os@OSs-MacBook-Pro/usr/local/bin/python3
```

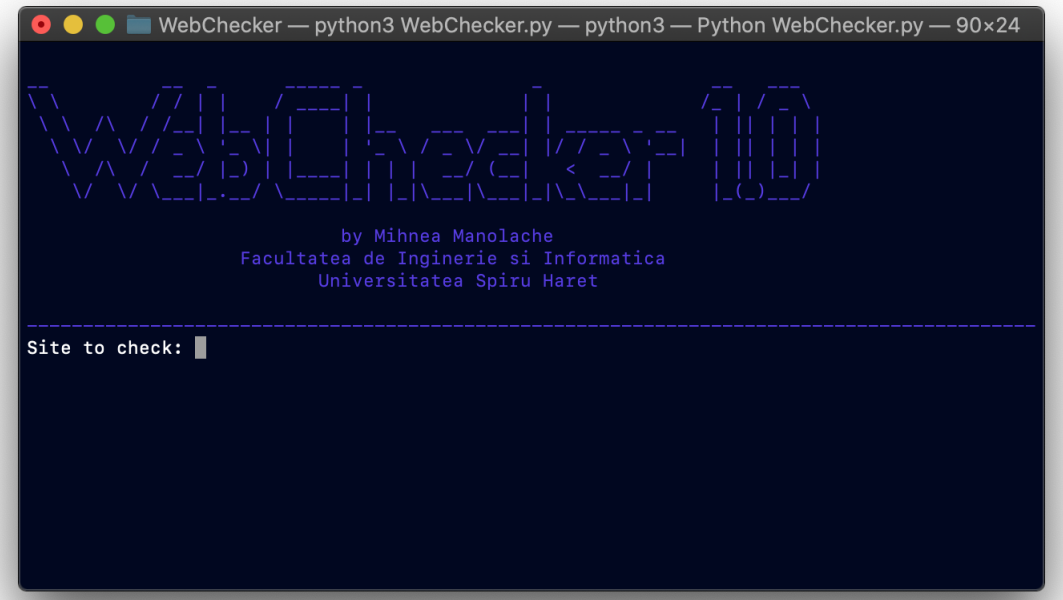
Execuția propriu-zisă se poate realiza în două moduri: fie rulând pur și simplu programul, fie specificând argumentele în comanda terminal.

4.1. Rularea directă a programului

Pentru a rula programul fără argumente, vom introduce în terminal comanda:

```
~/Desktop/WebChecker » python3 WebChecker.py
```

** Rularea aplicației necesită activarea mediului virtual*



Pentru a verifica statusul aplicației, vom introduce adresa URL a aplicației țintă. În exemplul următor, vom testa aplicația situatiescolara.spiruharet.ro.

După cum putem observa, programul ne returnează toate informațiile menționate anterior. Aplicația situatiescolara.spiruharet.ro este disponibilă, întoarce statusul HTTP 200, este găzduită pe un server Apache-Coyote/1.1, iar adresa IP a acestui server este 94.176.180.23.

4.2. Rularea programului cu argumente

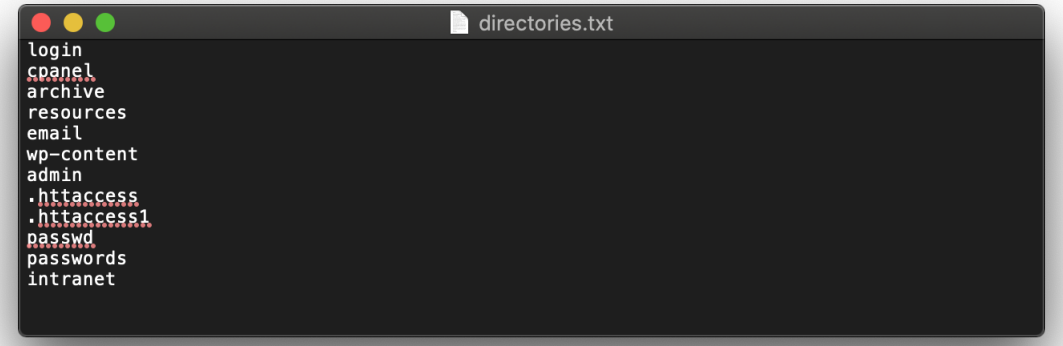
Pentru a rula programul cu argumente, vom introduce în terminal comanda, după care vom specifica argumentele. În acest sens, argumentul „-u”/„--url” este obligatoriu, iar argumentul „-b”/„--brute” este facultativ.

Execuția comenzii următoare va întoarce același rezultat ca exemplul anterior:

```
~/Desktop/WebChecker » python3 WebChecker.py -u
situatiescolara.spiruharet.ro
```

Dacă însă utilizatorul va specifica argumentul „-b”/„--brute”, programul va întoarce un rezultat diferit, deoarece va încerca să găsească directoare disponibile ale aplicației țintă.

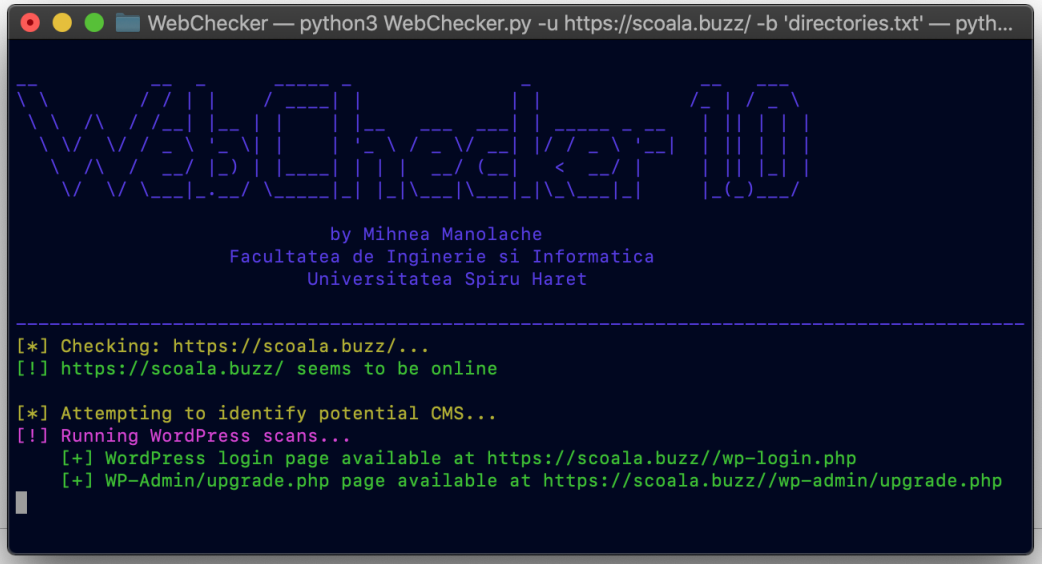
De asemenea, trebuie știut că introducerea argumentului „-b”/„--brute” trebuie urmată de destinația unui dicționar de cuvinte (ex. fișier .txt). Un exmplu de dicționar pentru brute-force este următorul:



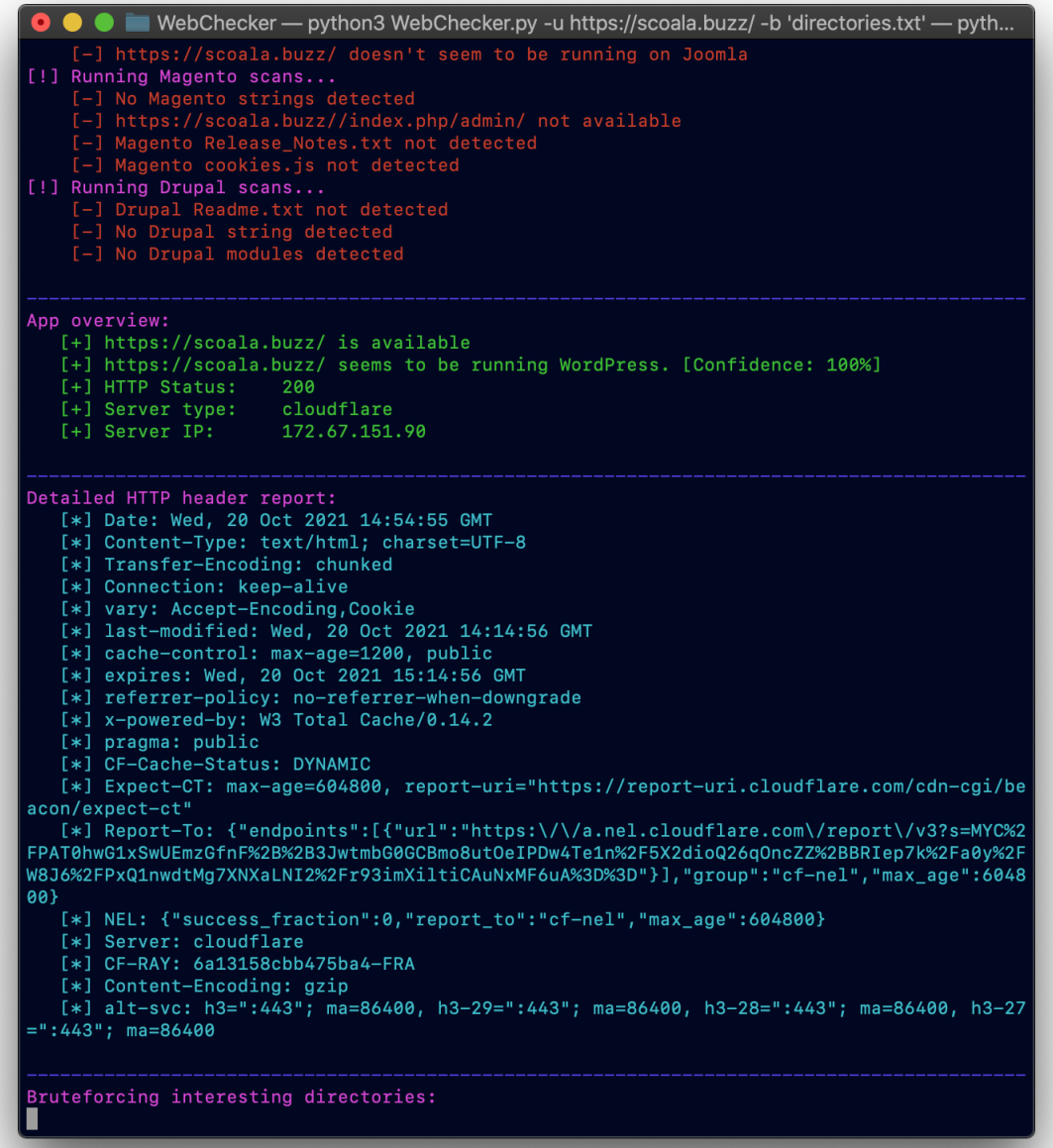
În exemplul următor, vom analiza o aplicație web care rulează pe platforma wordpress, respectiv website-ul unei școli de șoferi locale. De asemenea, vom utiliza argumentul „-b”/„--brute”, utilizând lista „directories.txt” pentru tehnica de brute-force a directoarelor. Pentru aceasta, vom introduce în terminal comanda:

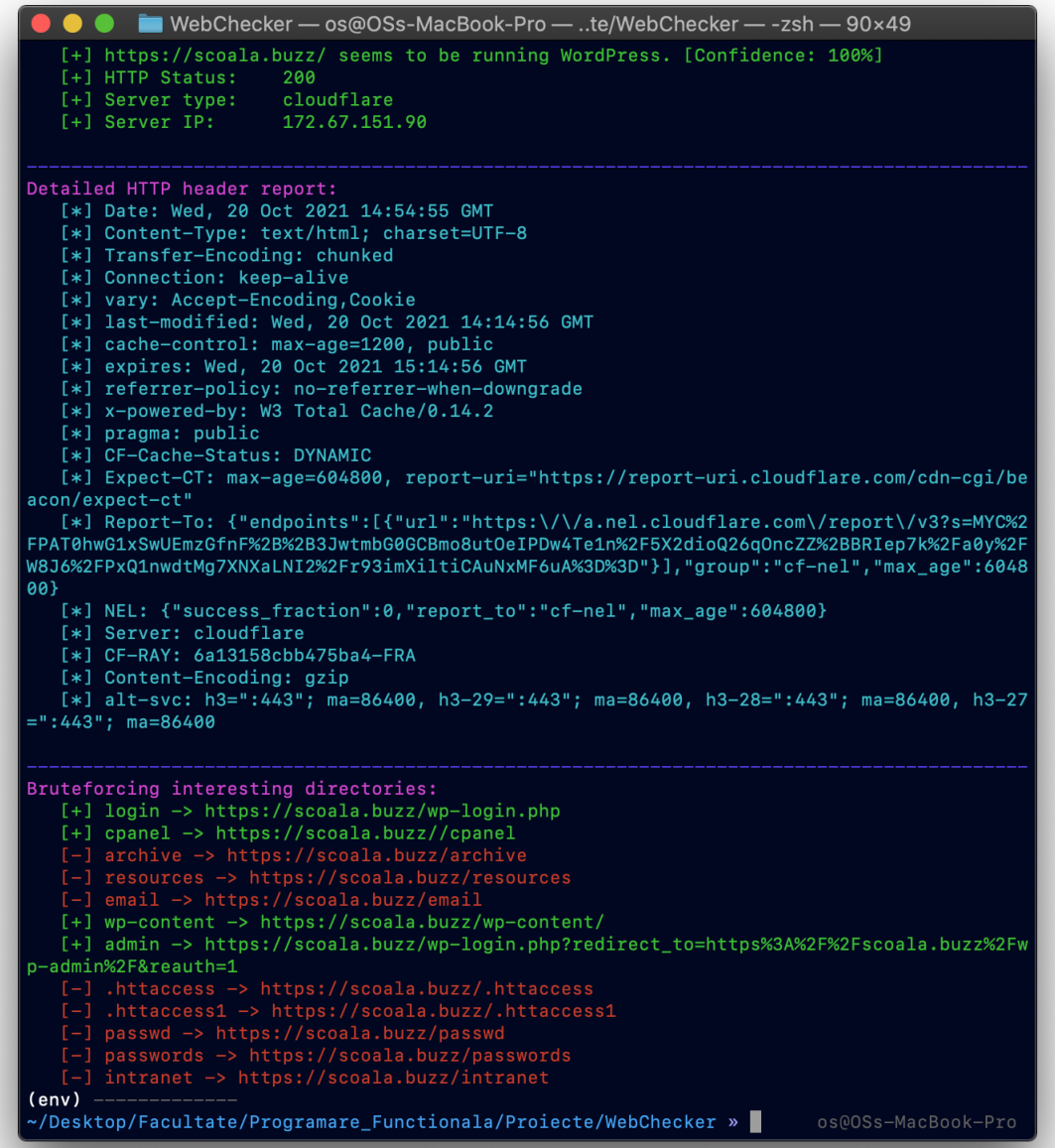
```
~/Desktop/WebChecker » python3 WebChecker.py -u
https://scoala.buzz/ -b 'directories.txt'
```

Observăm că programul nu solicită utilizatorului să introducă adresa URL a aplicației țintă:



De asemenea, pe lângă rezultatul inițial, programul va afișa utilizatorului rezultatele tehnicii brute-force, împreună cu URL-ul către care cuvântul din dicționar a redirecționat:




WebChecker — os@OSs-MacBook-Pro — ..te/WebChecker — -zsh — 90×49
[+] https://scoala.buzz/ seems to be running WordPress. [Confidence: 100%]
[+] HTTP Status: 200
[+] Server type: cloudflare
[+] Server IP: 172.67.151.90
Detailed HTTP header report:
[*] Date: Wed, 20 Oct 2021 14:54:55 GMT
[*] Content-Type: text/html; charset=UTF-8
[*] Transfer-Encoding: chunked
[*] Connection: keep-alive
[*] vary: Accept-Encoding,Cookie
[*] last-modified: Wed, 20 Oct 2021 14:14:56 GMT
[*] cache-control: max-age=1200, public
[*] expires: Wed, 20 Oct 2021 15:14:56 GMT
[*] referrer-policy: no-referrer-when-downgrade
[*] x-powered-by: W3 Total Cache/0.14.2
[*] pragma: public
[*] CF-Cache-Status: DYNAMIC
[*] Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
[*] Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?s=MYC%2FPAT0hwG1xSwUEmzGfnF%2B%2B3JwtmbG0GCBmo8utOeIPDw4Te1n%2F5X2dioQ26qOncZZ%2BBRIep7k%2Fa0y%2FW8J6%2FPxQ1nwdtMg7XNXaLNI2%2Fr93imXiltiCAuNxMF6uA%3D%3D"}],"group":"cf-nel","max_age":604800}
[*] NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
[*] Server: cloudflare
[*] CF-RAY: 6a13158cbb475ba4-FRA
[*] Content-Encoding: gzip
[*] alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400, h3-28=":443"; ma=86400, h3-27=":443"; ma=86400
Bruteforcing interesting directories:
[+] login -> https://scoala.buzz/wp-login.php
[+] cpanel -> https://scoala.buzz//cpanel
[-] archive -> https://scoala.buzz/archive
[-] resources -> https://scoala.buzz/resources
[-] email -> https://scoala.buzz/email
[+] wp-content -> https://scoala.buzz/wp-content/
[+] admin -> https://scoala.buzz/wp-login.php?redirect_to=https%3A%2F%2Fscoala.buzz%2Fwp-admin%2F&reauth=1
[-] .httaccess -> https://scoala.buzz/.httaccess
[-] .httaccess1 -> https://scoala.buzz/.httaccess1
[-] passwd -> https://scoala.buzz/passwd
[-] passwords -> https://scoala.buzz/passwords
[-] intranet -> https://scoala.buzz/intranet
(env)
~/Desktop/Facultate/Programare_Functionala/Proiecte/WebChecker »
os@OSs-MacBook-Pro