Branch: master hackerU_python / lessonPlan / lesson7_portScanner_webApplicatino.md Find file Copy path

t-0-m-1-3 added folders 190da8f on Jul 30, 2018

1 contributor

270 lines (180 sloc) 7.5 KB Raw Blame History

Intro To Python For Security

A port scanner is an application designed to probe a server or host for open ports. It's a great too for Admins to verify if their network is secure and following security policies.

It's also used by attackers to identify network services running on a host and exploit vulnerabilities.

A port scan or portscan, is a process that sends client requests to a range of server port addresses on a host. The goal is to find an active port;

Port Scans are integral parts of the information gathering stage of a campaign.

Exercises

1. Explain what the code below does
2. Change the code below in order to scan one port on a range of ips given by the user

Port Scanner

```
import socket
rhsot=raw_input()
sport=raw_input()
eport=raw_input()

for i in range(sport, eport):
    try:
        s = socket.socket(soccket.AF_INET,socket.SOCK_STREAM)
        s.settimeout(5)
        s.connect((rhost,i))
        print "\n Found Open Port: ", i
        s.close()
    except:
        pass
```

Exercises

Scan your network with the scanner and watch the trafic using wireshark

Attacking Web Applications

Fetching Web Pages

- The most basic function of going out and grabbing some data is hitting an http endpoint
- you can use the modules `urllib` , and `urllib2`
- These allow for argument encoding

You'll need Apache installed on linux

Fetch the pages

```
import urllib

httpResponse = urllib.urlopen("http://localhost")

httpResponse.code # this will hold the response code (404, 200, etc)

print httpResponse.read() #outputs the website html code
```

Take some time and play around with the methods inside of the modules

`dir(httpResponse)` will display a bunch of information about your variable

`httpResponse.url` will output the url `dir(httpResponse.headers)` will pull up more information about the response header `httpResponse.headers.items()` will get the information from the headers

To print the data in an easily readable way, use a for...in loop

```
for header, value httpResponse.headers.items():
    print header + ':" + value
```

You can also encode your arguments in webpages

```
url = "http://www.hackeru.co.il/index.php?act=landpage&id=21"
base_url = "http://www.hackeru.co.il/index.php"
args = {'act':'landpage','id':21}

encode_args = urllib.urlencode(args)

fp2 = urllib.urlopen(base_url + '?' + encode_args)

fp2.code # check for a 200 response

fp2.read() #download the html from the site

# you can use urllib.urlretrieve() to monitor the downloads progress from a page
```

`urlencode()` isn't really the best option when passing information because of how it handles special characters, you can checkout the man page for `quote()` and `quote_plus()` for how they can help

Parsing from On the Web:

Web data primarily comes in a few forms: `HTML` , `XHTML` , `XML` , and `JSON` . We need to have some ways to parse all of these different forms, and how to parse the data they offer to us. We also need a way to send this data across the wire when it is requested.

Parsing HTML:

HTML is a heriarchical data structure, so iterating through the series of HTML tags that you will encounter can be a tedious task. Thankfully, the python community offers a wide array of tools out there to take much of the burden off your shoulders.

- `LXML`
- `BeautifulSoup`
- `HTMLParser`
- and many more

Most websites have terribly implemented HTML code...

Beautiful Soup:

- Version 4 onwards allows use of `lxml` and `html5lib` tend to handle bad `HTML` better

Parser Comparison

|Parser | Typical Usage | Advantages | Disadvantages | | | | | | Python HTML parser
| `BeautifulSoup(markup,"html.parse")` |

- Batteries included
- speed is moderate
- flexible

|
- previous versions not as useful

| | `lxml` 'sHTML parser | `BeautifulSoup(markup,"lxml")` |
- Very fast
- Flexible

|
- dependent on `C`

| | `lxml` 'sHTML parser | `BeautifulSoup(markup,"lxml","xml")` or `BeautifulSoup(markup,"xml")` |
- Very fast
- only supported XML parser

|
- dependent on `C`

| | `html5lib` | `BeautifulSoup(markup,html5lib)` |
- extremely flexible
- parses web pages like a browser
- Creates valid HTML5

|
- dependent on `python2`
- slow

|

Parsin with Beautiful Soup:

```
import urllib
```

```
from bs4 import BeautifulSoup

html = urllib.urlopen("http://www.hackeru.co.il/courses/index")

html.code #response 200?

#beautiful soup uses html parser by default, we can switch it to LXML

bt = BeautifulSoup(html,"lxml")

bt.title #will show the title of the page

bt.title.string # will print unicode

bt.meta # will fetch the meta tags
bt.meta.next #fetches the next tag
bt.meta.next.next #fetches the next tag

#more pythonic way

allMetaTags = bt.find_all('meta')

allMetaTags[0] #display the first tag, or you can fetch the internals with a key

allMetaTags[0]['content']
```

Exercises:

1. Create a snippet that will strip a webpage of links

`bt.get_text()` # will strip text from a page

2. What other ways does beautiful soup have to iterate through tags?

Screen Scraper:

Dependent on the HTML Structure, slight coding changes can throw off the output.

Strip Links:

```
videoLink = bs.find('iframe',{'title' : 'YouTube video Player'})
```

You can get the actual link using `videoLink['src']` to pull the source tag

Exercises:

1. Find 5 web pages you want to pull information from and code some parsers

Mechnaize Library:

Based on a `Perl` module `WWW:Mechanize` The library allows for stateful programming and browser emulation. A lot of power, built into a teeny tiny module.

```
import mechanize

br = machnize.Browser()

br.open('http://www.hackeru.co.il/index.php')

for form in br.forms():
    print form
```

Exercises:

- are there better ways to check for hidden fields? `help('bs4')`
- Install `DVWA` on kali
- Try `SQL-I` on the form fields and check which fields are vulnerable to SQL injection.
- Check out the `SQLMap source code`

On The Web

Mechanize handles cookies by itself; it understands how to *browse* an application

with Mechnaize you can click links, fill and submit forms, maintian state, and navigate the web.

```
import mechnize

br = mechanize.Browser()
br.open('http://moodle.hackeru.co.il/login')
# with the page loaded we can now check for forms

 #is this the most pythonic way??
for form in br.forms():
    print form

br.select_form(nr=0)
    #we're selecting the first form
br.form['username']='david'
br.form['password']='P@$$w0rd'
#set the initial values in the variables we want to submit into the form.

print br.response().read() # check the servers response to the submission

for link in br.links():
    print link.url + ':' + link.text
# with this for loop we can check for the text of any links

new_link = br.click_link(text="hackeru[IMG] ChanPassword")
# grab the browser link and click it

br.open(new_link)

print br.response().read()

for form in br.forms():
    print form
```

How about we try using this on google??

```
br.open('https://gmail.com')

print br.response().read()

for form in br.forms():
    print form
```

`<TextControl(Email=)>;<PasswordControl(Passwd)>` are the fields you need to look for and fill in.

Exercises:

1. Create a script that will log into the Gmail
2. Check out `help('mechanize.CookieJar')` - what's some of the things you can do with it?