#NullHyd

Python for Web Security

Code Warriors

Sanjeev Jaiswal (Jassi)(2019)

author_profile

```
{
“Name”                : “Sanjeev Jaiswal”
“Nickname”            : “Jassi”
“Twitter_handle” : “jassics”
“Mail_id”             : “jassics[at]gmail[dot]com”
“Skills”              : [“AppSec”, “AWS Security”, “Perl”, “Python”]
“Interests”           : {
                          “Learning”          : “AWS Security Automation”,
                          “Want_to_learn” : “Security Automation in DevOps pipeline”
                        }
}
```

(“My”, “Assumptions”)

- You have python and pip installed in your machine
- You can execute python scripts
- You can install dependent libraries using pip
- You can write basics of python program
- You can understand python script
- You understand web security basics
- You know how request response of a web url works
- Burp is installed and configured to get proxy request

[List of Items]

- Why Python
- Fundamentals of Python (Minimal)
- Quick walk through of existing scripts
- Writing few web security scripts
- Creating minimal Burp plugin
- Resources for Learning
- What's Next

Why Python

- Easy to learn and Clean Syntax Code
- Widely being used in security domain
- Open- Source and Vast Community Support
- Automated memory management
- Support to Glue for other languages
- Good for quick and dirty jobs in security ;)
- Lots of security tools available in Python \m/

Fundamentals of Python (quick tour)

- Variable names
- Numbers and Strings
- Basic Operators
- Loops and statements
- Functions
- Data Structures i.e. Lists, Tuples, Dictionary
- File handling
- Regular Expression glimpse
- Modules

Python Fundamentals

Variable Names

- Make it human readable
- Keep it short but descriptive
- Don’t start with number
- Don’t use $ in your variable name
- all_lowercase_with_underscore
- Advisable not to start with underscore i.e. _var_name
- Also not advisable to use var.__name or __var_name
- Variable types depend upon the data being used.

Numbers and Strings

- Integer: 2, 4, -13, 130
- Float: 2.41564, 2.45, -1.2, 97.99
- String: ‘Apple’, “Hello, How are you doing?”
- int() for integer
- float() for float
- str() for string

Basic Operators

That operates pythonic way

Arithmetic Operators

- + : used for addition of two digits
- - : used for subtraction of two numbers
- * : used for multiplication
- / : used for division, you will get results in float
- //: floor division, you will get the quotient
- %: modulo division, you will get remainder as a result

Relational Operators

- > : greater than
- < : less than
- == : equals to
- != : not equal to
- >= : greater than equal to
- <= : less than equal to

Logical Operators

- and : returns True if both statements are true
- or : returns True if any of the statements are true
- not : returns False if the result is true

Bitwise Operators

- ~ : inverts all the bits
- | : sets bits to 1 if one of the bits is 1
- ^ : sets each bit to 1 only one of two bits are 1
- & : sets each bit to 1 if both bits are 1
- >> : signed right shift
- << : left shift

Assignment Operators

- = : assigns right side value to left side variable
- += : x+=3 means x = x+3
- -= : x-=3 means x = x-3
- *= : x*=5 means x = x*5
- /= : x/=4 means x = x/4
- //= : x//7 means x = x//7
- %= : x%=2 means x = x%2
- **= : x**=2 means x = x**2

Data Structure

List, Tuples, Dictionary

Lists

- Most important and widely used in python
- Its like array in C
- It is accessed by using index number
- Index number starts with 0
- You can use list methods like pop, append, insert, extend, sum, len, min, max, sort, reverse, del, remove, clear
- Syntax:
  List_variable_name = [] #empty list
  List_with_value = [21, 25, 80, ‘ssl’, ‘web server’, 21, 80, 8080, 21]

Tuples

- Similar to list
- But immutable
- Denoted by () where as list is denoted by []
- You can use count and index
- You can join two tuples
- You cannot insert, delete
- (‘single item’,) is a tuple, but (‘single item’) is a string
- only_allowed_ports = (80, 443, 8000, 8080, 9001)

Dictionary

- When you need key value pair like port number with port service name
- Dictionary_var = {“key” : value}
- Value can be any object, it can be even another dictionary
- dict_var = dict() or dict_var = {} # for empty dictionary
- port_service = {“ssh”: 22, “ftp”: 21, “http”: 80}
- for service in port_service:
  print(“{} : {}”.format(service, port_service[service]))
- port_service.pop(“http”)
- port_service.update({“ssl”: 443})
- port_with_service = port_service.items()
- print(port_with_service)

Loops

Let’s iterate over items

for loop

for iterator in sequence:
    statement(s)

```
Example 1
print("Port List Iteration")
ports = [22, 25, 80, 443]

for port in ports:
    print(port)
```

```
Example 2
print("Dictionary Iteration")

port_dict = {
    'Ftp' = 21
    'ssh' = 22
    'Smtp' = 25
    'Https' = 443
}

for service in port_dict :
    print("%s : %d" %(service,
port_dict[service]))
```

while loop

while expression:
    statement(s)

while expression:
    statement(s)
else:
    Some other
statement(s)

```
Example 1
x = 10
while x>0:
    print(x)
    x--

Example 2
x = 0
while x<10:
    x += 1
    print(x)
else:
    print('Came out of the
loop')
```

Statements

Decision Making Statement

- if statement
- if else statement
- If elif else ladder
- Nested if else

Loop Control Statements

- break
- pass
- continue

Let's write functions

- Reusable block of codes
- Can pass arguments to it
- Built-in functions: print(), len(), sorted(), int(), str() ...
- Structure:
  - Start with keyword def and then function name with parameters inside parenthesis as optional
  - Ex: def print_status_code(url):
  - It can have return statement
- Call named function with function name. `Ex: print_status_code(url)`

```
import requests

url = 'https://opsecx.com'

def get_status_code(url):
    response = requests.get(url)
    return response.status_code

status_code = get_status_code(url)

print (status_code)
```

File Handling

- Python provided an in-built function to read and write files
- Syntax:
  - file_input = open(“file_name”, “mode”)
  - Mode can be r for read, w for write, a for append
  - + suffix to mode means, create if it doesn’t exists
  - Ex: `file_input = open(“ip-ranges.json”, w+)`
  - Write to file_input. `file_input.write(“127.0.0.1\n”)`
  - Close the file. `file_input.close()`
- Other modes are x for creation, b for binary (t for text, by default)
- Read the file content by using read(chars), readline() or readlines()
- Another way to read file is `with open(...) as file_input:`

```
for line in file_input:
    # do something with that line
```

Regular Expression

- Learn regex basics from here: https://www.regular-expressions.info/quickstart.html
- import re # import regular expression module in Python
- Python module re provides full support for Perl-like regular expressions in Python
- re.match(), re.search(), re.sub()
- Modifiers or optional flags in re: re.S, re.M, re.I etc.
- Get the matched content using group(num) or groups()
- You can compile the pattern as well. re.compile(pattern)
- `matched = re.search(pattern, string)` ie equivalent to

```
comp_pattern = re.compile(pattern)
matched = comp_pattern.match(string)
```

Let's use Modules

- Think it like a library
- Just an another file with python codes
- It can define classes, functions, variables etc
- Use any of the below way to use python module:
  - `import module_name`
  - `import module1, module2, module3, moduleN`
  - `import some_module as your_convenient_name`
  - `from module1 import *`
  - `from module_name import <specific>`

Commonly used modules in security

- re : used for regular expression
- os, sys, socket : system based calls
- requests, webbrowser, wappalyzer, urllib3, pyautogui : website manipulation
- json, csv, xml
- from scapy.all import *
- from ftplib import FTP
- from faker import Faker
- from bs4 import bs
- nmap, dns, whois, ipaddress
- pip install pycrypto, hashlib, base64
- Anything else? Please add here

walkthrough_existing_security_scripts

- Python wappalyzer
- Gittyleaks or trufflehog
- KickS3 orS3 Recon
- Analyse Security Headers
- Bruteforce Login or Instagram Account Bruteforce
- Scapy
- Get git user details

Writing Scripts

Get hands dirty (Git repo)

#1 Fetch request and response headers of a url

1. url is provided
2. Use requests modules in your script
3. Instantiate a requests object
4. Use get method on url using that created object as in #3
5. Save request and response headers in variables respectively
6. Loop through request and response headers respectively and
7. Print the header contents

#2 Filter request and response headers of a url

1. url is provided
2. Use requests modules in your script
3. Instantiate a request object
4. Use get method on url using a request object
5. Save response headers in variable
6. Create a list of filtered header contents
7. Loop through response headers and
8. Check if that header is in our filtered header content
9. Print the header content, if it's in filtered response header

#3 [TRY AT HOME]: Scan and Report Security headers gap

- Simulate in cli: https://securityheaders.com/?q=null.co.in&followRedirects=on
- Use requests and colorama module
- Get the response headers
- Look for security headers that exists in your suggested security headers dictionary
- If missing, print in red color with the suggestions saved in dictionary against that header
- If security header found
  - Print in green color, if header is implemented properly
  - Print in red color, if that security header is missing some implementations
- Once done all, you can give your own rating based on your rating calculator (Future)

Smart Password Generator based on victim's details (Project?)

- There are many password files available, but generic
- How about generating passwords based on these combination:
  - Name (firstname, lastname, nickname etc.)
  - Hobbies
  - Locations
  - Favorite (Food, pet, movie, family members etc)
  - Date of [Birth, marriage, family's other dates)
  - Profession
  - And password character ranges
- Either gather the information and pass to the script as an input file
- Or Ask all those questions through cli
- And finally generate possible passwords and output as victim_password.txt

Writing Custom Burp Plugin

Basic Burp Extension using Python: Part 1

- It is based on https://portswigger.net/burp/extender/writing-your-first-burp-suite-extension
- Create a directory to store your extensions scripts, we will use this directory now onwards for our extension
- Download Jython standalone (.jar) from here: https://www.jython.org/download.html
- Keep this jar file in the same extension directory for convenience
- Configure Burp to use Jython. Extender -> Options -> Python Environment
- Create a python script and import necessary modules in that script
- Write BurpExtender Class i.e. class BurpExtender(IBurpExtender):
- Add your custom written Python script in Burp Extension
- Check for output or any error in popup window

Basic Burp Extension using Python: Part 2

- Once Burp extension Part 1 exercise is successful
- Add Own custom tab in HTTP request
- Show headers just like request headers in custom tab
- You would need to use
  - from burp import IMessageEditorTabFacory
  - from burp import IMessageEditorTab
- Add IMessageEditorTabFactory in class
- Get helpers object: self._helpers = callbacks.getHelpers()
- Register the object: callbacks.registerMessageEditorTabFactory
- Create a new instance to DisplayValues
- Define DisplayValues class with all the essential functions
- Verify if it's showing the tab and contents within that tab

Basic Burp Extension using Python: Part 3 (Try at Home)

- Once Burp extension Part 2 exercise is successful
- Create similar tab under Proxy->HTTP History->Response tab
- Loop through the response header contents
- Create a dictionary of security header with suggestions
- Look for security headers and
  - Show present or absent based on header presence compared with your list
  - Show if present security header is implemented properly
- Note: You might need to go through few of the Burp Extender APIs Code snippets

Learning Resources

- Violent Python
- Black hat Python
- Automating boring stuffs
- Python for Pentesters by Vivek Ramachandran
- Python for everybody specialization (Coursera)
- Black Hat Python for Pentesters and Hackers - Video (PackT)
- Cracking Codes with Python

What's Next

- Python for Network Security
- Python for Security Automation
- Exploits in Python
- Secure Coding in Python
- AI/ML in Cybersecurity using Python
- Python for Crypto
- Malware Analysis using Python
- Forensics using Python
- And many more

Follow for more contents...

Twitter: https://www.twitter.com/jassics

Linkedin: https://www.linkedin.com/in/jassics/

Github: https://github.com/jassics

Youtube: https://www.youtube.com/c/flexmind

Newsletter: https://www.getrevue.co/profile/jassics