



Python Basics for Web App Pentesters

Justin Searle
Senior Security Analyst - InGuardians Inc.
justin@inguardians.com

Why Python



- Pre-installed on Mac and Linux
- Easy to install on Windows
- Easy to write scripts that run on all OSes
- Easy to read and collaborate
- Very complete set of standard libraries
- Many stable and powerful 3rd party libraries

Python Shell



- Using an interactive python shell
 - type “python” on your command line
 - type python commands
 - they execute when you hit enter
- Why use the shell?
 - Easy way to learn the language
 - Great way to debug portions of code
 - Nice for PoC functions and loops
- Beyond the basic shell
 - Consider ipython or IDLE after you get your feet wet
 - Provide richer functionality and productivity

Input and Output



```
print('This site is protected by SSL.')
```

Basic output

```
answer = raw_input('Do you wish to continue? ')
```

Basic input

object oriented bliss

```
if answer.lower() == 'no':  
    print('Enjoy the rabbit hole!')  
else:  
    print('Here is your blue pill')
```

if / then / else
conditional
statements. Notice
the colons followed
by mandatory line
indentations

Simple HTTP Requests



The library that does the magic

```
import urllib2
```

This doesn't make the request, it simply packages the request

```
request = urllib2.Request('http://www.inguardians.com')
```

Don't forget the "http://"

```
response = urllib2.urlopen(request)  
payload = response.read()
```

This sends the request, catches the response, and extracts out the response payload

```
print(payload)
```

POST Requests



```
import urllib2, urllib
```

```
url = 'http://whois.arin.net/ui/query.do'
```

```
data = { 'flushCache' : 'false',  
         'queryinput' : '198.60.22.2'}
```

```
data = urllib.urlencode(data)
```

```
request = urllib2.Request(url, data)
```

```
response = urllib2.urlopen(request)
```

```
payload = response.read()
```

```
print(payload)
```

Add your POST
data to a dictionary

Then urlencode your data
(don't forget to import urllib)

If you provide urllib2
with request data, it will
assume a POST

Working with Headers



```
import urllib2
```

```
url = 'http://google.com/#q=samurai-wtf'
```

```
headers = { 'User-Agent' : 'Mozilla/5.0 (iPhone)' }
```

```
request = urllib2.Request(url, None, headers)
```

```
response = urllib2.urlopen(request)
```

```
headers = response.headers
```

```
print(headers)
```

Add your headers to a dictionary

If you are doing a GET, use None for data

Writing to a File



```
import urllib2
```

```
request = urllib2.Request('http://inguardians.com/info')
```

```
response = urllib2.urlopen(request)
```

```
payload = response.read()
```

```
with open('info.html', 'wb') as file:
```

```
    file.write(payload)
```

Try opening a file, in
write and binary modes

Write the payload to the file

Filtering Responses



```
import urllib2, re
request = urllib2.Request('http://inguardians.com/info')
response = urllib2.urlopen(request)
payload = response.read()
```

Build your regex
using a raw string,
grouping desired text

```
regex = r'<dt class="title">(.*?)</dt>'
results = re.findall( regex , payload )
```

Search payload
for all instances
of your regex

```
for result in results:
    print(result)
```

Loop through your results printing them

Basic Authentication



```
import urllib2
```

Setup needed variables

```
url = 'http://somesite.com/'
```

```
username = 'admin'
```

```
password = 'letmein'
```

Setup password manager

```
password_mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()
```

```
password_mgr.add_password(None, url, username, password)
```

Add passwords

```
authhandler = urllib2.HTTPBasicAuthHandler(password_mgr)
```

```
opener = urllib2.build_opener(authhandler)
```

Connect handler

```
urllib2.install_opener(opener)
```

```
response = urllib2.urlopen(url)
```

```
payload = response.read()
```

```
print( payload )
```

Build and install so all requests automatically use the password manager

A Tale of Two Libraries



urllib2

HTTP, HTTPS, & FTP
Auto Parses URI
Follows Redirections
Uses a Cookie Jar
Auth: Basic & Digest
Methods: GET & POST
Supports Proxy Servers
Auto Closes Connections

httplib

HTTP & HTTPS
No URI Parsing
Doesn't Follow Redirects
Doesn't Store Cookies
Authentication: None
Method: Any
No Proxy Support
Manually Close Connection

Using httplib



```
import httplib
```

Create a “connection” object

```
connection = httplib.HTTPConnection("python.org")  
connection.request("TRACE", "/index.html")
```

```
response = connection.getresponse()  
payload = response.read()
```

Actual request
made here

```
print(payload)
```

Read response
and extract
payload

Fuzzing and Brute Force



```
import urllib2, re
```

```
with open('list.txt', 'rb') as f:  
    list = f.readlines()
```

Read file as a list of values, 1 per line. Like 1533095958

Convert to string and remove newline

```
for item in list:
```

```
    url = 'http://www.facebook.com/people/a/' + str( item.strip() )
```

```
    try:
```

```
        response = urllib2.urlopen(url, None, 1)
```

```
    except IOError:
```

```
        pass
```

```
    else:
```

```
        payload = response.read()
```

```
        regex = r"canonical" href="(.*)"
```

```
        match = re.search(regex, payload)
```

```
        if match: result = match.groups()
```

```
        print( result[0] )
```

Prevent missing pages from throwing an error and stopping the script

Find first match and if found, print to screen

pyCIT



- Python Commandline Interface Templates
 - <http://code.google.com/p/pycit>
 - a collection of python templates for creating command line tools
 - great tool for beginners to show how to do the basics
 - saves advanced users time by providing the basic and much more
- Each templates will give you:
 - Built in command line arguments, easily modifiable
 - Provides a help page if no arguments are given
 - Tracks and displays your script version
 - Verbosity and debug functions with command line flags
 - Command line options and functions for reading and writing to files

pyCIT Templates



- Completed Templates
 - Basic file read/write access
 - Single-threaded http requests (basic wget/curl functions)
- Planned Templates
 - Binary file read/write access with hex decode (basic xxd/hexdump functions)
 - Multi-threaded http requests (basic wget/curl functions)
 - Raw socket client and service (basic netcat functions)
 - Raw usb device access
 - Interactive CLI interface

Contact Information



Justin Searle
Senior Security Analyst - InGuardians

justin@inguardians
justin@meeas.com
twitter: @meeas

<http://code.google.com/p/pycit>
<http://samurai.inguardians.com>