Python Modules for InfoSec

Respondo

Who am I

Jonathan Respeto

I work at Akamai Technologies as a Security Intelligence Response Team Engineer.

What do I do:

Research technologies that can be misused to attack our customers and created defences for those attacks before they happen.

Request

Though not exclusively for security, the Requests library is frequently used in web security and penetration testing scripts for making HTTP requests easier and more human-friendly.

- Making request to APIs
- Automating Exploits
- Scraping for Threat Intelligence

Request - example

```
import requests

# The URL to which the request will be sent
url = "https://httpbin.org/get"

# Custom headers to be sent with the request
headers = {
    "User-Agent": "MyApp/1.0",
    "Accept": "application/json"
}

response = requests.get(url, headers=headers)

# Checking if the request was successful
if response.status_code == 200:
    response_json = response.json()

    print("Response from server:")
    print(response_json)
else:
    print(f"Failed, status code: {response.status_code}")
```

```
import requests

# Create a session object
session = requests.Session()

# Define custom headers for the session
session.headers.update({
    "User-Agent": "MySessionApp/1.0",
    "Accept": "application/json"
})

# URL for the first request
url1 = "https://httpbin.org/cookies/set/sessioncookie/123456789"

# URL for the second request to test if the cookie has been set
url2 = "https://httpbin.org/cookies"

# Making the first request to set a cookie
response1 = session.get(url1)

# Checking if the first request was successful
if response1.status_code == 200:
    print("First request successful, session cookie set.")
else:
    print(f"Failed, status code: {response1.status_code}")

# Making the second request to check the cookie
response2 = session.get(url2)

# Checking if the second request was successful
if response2.status_code == 200:
    print("Second request successful. Cookies received:")
    print(response2.json())
else:
    print(f"Failed, status code: {response2.status_code}")
```

Selenium

Selenium is not primarily an InfoSec module but a portable framework for testing web applications. It provides a playback tool for authoring functional tests without the need to learn a test scripting language (Selenium IDE).

Some InfoSec uses:

- Web Scraping for Intelligence Gathering
- Web Application Penetration Testing

Selenium is a powerful tool for automating web browsers. i.e Chrome/Firefox…

Selenium - example

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from time import sleep

# Specify the path to your WebDriver executable if it's not in your PATH
# driver_path = '/path/to/your/webdriver'

# Initialize the WebDriver (in this case, Chrome)
# If your WebDriver is in the PATH, you can initialize without specifying the path
driver = webdriver.Chrome()

try:
    # Open the URL
    driver.get("https://httpbin.org/user-agent")

    # Wait for the page to load
    sleep(2)

    # Locate the element containing the user-agent string
    # Adjust the selector based on the actual page structure you're scraping
    user_agent_element = driver.find_element(By.XPATH, '//pre')

    # Print the text found in the element
    print("Found user-agent string:", user_agent_element.text)

finally:
    # Close the browser window
    driver.quit()
```

Scapy

A powerful Python-based interactive packet manipulation program and library. It can forge or decode packets of a wide number of protocols, send them on the wire, capture them, and match requests and replies.

- Packet Crafting and Manipulation
- Packet Sniffing
- Network Discovery and Monitoring - port scanning
- Protocol Dissection and Analysis
- Visualization Tools

Scapy - example - Making and Sending a Packet

```
from scapy.all import *

# Create an ICMP Echo Request packet destined for example.com
packet = IP(dst="example.com")/ICMP()

# Send the packet and receive the reply
reply = sr1(packet, timeout=2)

# Print the reply
if reply:
    reply.show()
else:
    print("No reply received")
```

Scapy - example - Sniffing and Filtering for a Packet

```
from scapy.all import *

# Define a packet processing function
def process_packet(packet):
    if ICMP in packet:
        print(f"ICMP packet from {packet['IP'].src} to
{packet['IP'].dst}")

# Start sniffing for ICMP packets
sniff(filter="icmp", prn=process_packet, count=10)
```

Scapy - example - Packet Scanning

```
from scapy.all import *

# Target IP address and port range
target_ip = "some_test_host.com_that_we_are_allowed_to_test"
port_range = [22, 80, 443]

# Perform a TCP SYN scan
answered, unanswered = sr(IP(dst=target_ip)/TCP(dport=port_range, flags="S"),
timeout=1, verbose=False)

# Check for ports that responded with a SYN-ACK (open ports)
for sent, received in answered:
    if received[TCP].flags == "SA":  # SYN-ACK means open port
        print(f"Port {sent[TCP].dport} is open")
```

Scapy - example - Printing a Graph of the Packet

```
from scapy.all import *

# Create an ICMP packet
packet = IP(dst="8.8.8.8")/ICMP()

# Generate a PDF file of the packet's structure
packet.pdfdump("icmp_packet_graph.pdf")
```

Scapy - Printing a Graph of the Packet

```python
from scapy.all import sniff
from graphviz import Digraph

def generate_mind_map(packets, filename='packet_mind_map'):
    """
    Generates a mind map from captured packets using Graphviz.

    :param packets: The list of packets to be included in the mind map.
    :param filename: The filename for the output graph (without extension).
    """
    dot = Digraph(comment='Packet Mind Map', format='pdf')

    # Set graph attributes for aesthetics
    dot.attr('node', shape='ellipse')
    dot.attr('graph', rankdir='LR')

    for packet in packets:
        # print(packet.summary())
        # print(packet.show())
        # print(packet['IP'])
        if 'IP' in packet:
            src = packet['IP'].src
            dst = packet['IP'].dst
            protocol = packet.sprintf("%IP.proto%")
            edge_label = f"{protocol}"

            # Add nodes and edges with protocol as label
            dot.node(src, src)
            dot.node(dst, dst)
            dot.edge(src, dst, label=edge_label)

    # Save and render the graph to a file
    dot.render(filename, view=True)

# Capture packets
print("Capturing packets... Please wait.")
# sniff as root user
packets = sniff(count=50, iface='en7', filter='ip')

# Generate mind map from captured packets
generate_mind_map(packets)
print("Mind map has been generated.")
```

```python
from scapy.all import *
import matplotlib.pyplot as plt
from collections import Counter

# Sniff packets
packets = sniff(count=10, iface='en7', filter='ip')

# Count occurrences of each protocol
protocols = [packet.proto for packet in packets if packet.haslayer('IP')]
protocol_counts = Counter(protocols)

# Prepare data for plotting
labels = [str(proto) for proto in protocol_counts.keys()]
sizes = protocol_counts.values()

# Plot
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Protocol Usage Distribution')
plt.show()
```

Python-nmap

python-nmap is library that allows Python developers to programmatically access and manipulate Nmap functionalities.

- Network Scanning
- Port Scanning
- OS and Software Version Detection
- Flexible Output Parsing

You need to have nmap installed.

python-nmap - example - Basic Network Scan

```
import nmap

# Initialize Nmap PortScanner
nm = nmap.PortScanner()

# Scan a subnet for hosts
nm.scan(hosts='192.168.1.0/24', arguments='-sn')

# Print the list of hosts that were up
for host in nm.all_hosts():
    print(f'Host {host} is {nm[host].state()}')
```

python-nmap - example - Port Scanning

```
import nmap

nm = nmap.PortScanner()

# Scan example.com for open ports in the range 22-443
nm.scan('example.com', '22-443')

# Print scan results for each host
for host in nm.all_hosts():
    print(f'Host : {host} ({nm[host].hostname()})')
    print(f'State : {nm[host].state()}')
    for proto in nm[host].all_protocols():
        print(f'----------\nProtocol : {proto}')

        lport = nm[host][proto].keys()
        for port in lport:
            print(f'port : {port}\tstate : {nm[host][proto][port]["state"]}')
```

python-nmap - Detecting OS and Service Version

```
import nmap

nm = nmap.PortScanner()

# Perform an in-depth scan to detect OS and service versions
nm.scan('example.com', arguments='-A')

for host in nm.all_hosts():
    print(f'Host : {host} ({nm[host].hostname()})')
    print(f'State : {nm[host].state()}')

    # OS detection
    if 'osclass' in nm[host]:
        for osclass in nm[host]['osclass']:
            print(f'OS Type : {osclass["osfamily"]} {osclass["osgen"]} ({osclass["type"]})')

    # Service version detection
    for proto in nm[host].all_protocols():
        print(f'Protocol : {proto}')
        lport = nm[host][proto].keys()
        for port in lport:
            print(f'port : {port}\tname : {nm[host][proto][port]["name"]}\tstate : {nm[host][proto][port]["state"]}\tproduct :
{nm[host][proto][port]["product"]}\tversion : {nm[host][proto][port]["version"]}')
```

python-nmap - example - Asynchronous Scanning

```
import nmap
import sys

# Callback function to handle scan result
def callback_result(host, scan_result):
    print(f'--------------\nScan result for host: {host}')
    print(scan_result)

nm = nmap.PortScannerAsync()

try:
    # Start an asynchronous scan
    nm.scan(hosts='192.168.1.0/24', arguments='-sP', callback=callback_result)

    # Wait for the scan to complete
    while nm.still_scanning():
        print("Scanning...")
        nm.wait(2)  # Wait 2 seconds and check again
except KeyboardInterrupt:
    print("Scan aborted by user")
    sys.exit(0)
```

Volatility

Volatility is an advanced memory forensics framework designed for the analysis of volatile memory (RAM) captures. It's widely used in the field of digital forensics and incident response to investigate memory artifacts from various operating systems.

Volatility - example - Identifying Linux Kernel Version

```
import volatility3.framework as v3f
from volatility3 import framework, plugins
from volatility3.framework import contexts, interfaces, symbols
from volatility3.framework import automagic, constants, contexts, configuration
from volatility3.framework.configuration import requirements
from volatility3.framework import symbols
from volatility3.cli import text_renderer
from volatility3 import plugins

from volatility3.plugins.linux import pslist

# Initialize the context
ctx = contexts.Context()
# Input file in URI format
single_location = "file:///Users/jonathan/Downloads/MemoryDump_Lab1.raw"
ctx.config["automagic.LayerStacker.single_location"] = single_location
ctx.config["single_location"] = single_location
ctx.config['automagic.magic_plugins'] = ['linux.bash_history', 'linux.pslist.PsList']

plugins.__path__ = v3f.constants.PLUGINS_PATH
v3f.import_files(plugins, True)

plugin_list = framework.list_plugins()
# print(plugin_list)

plugin_name = "linux.pslist.PsList"

print(dir(ctx))
"""Use automagic to initialise volatility plugin configuration"""
plugin = plugin_list[plugin_name]
available_automagics = framework.automagic.available(ctx)
framework.automagic.choose_automagic(available_automagics, plugin)
framework.automagic.run(available_automagics, ctx, plugin, interfaces.configuration.path_join("plugins", plugin.__name__))

print('plugins.'+plugin_name)
print(ctx.config)
```

https://github.com/pinesol93/MemoryForensicSamples
https://github.com/volatilityfoundation/volatility/wiki/Memory-Samples
https://github.com/stuxnet999/MemLabs

And the list goes on and on….

- volatility3
- yara
- Impacket
- SQLAlchemy
- Beautiful Soup & lxml
- Twisted
- Flask
- subprocess
- asyncio…