Code Review Worksheet
CPSC 491/491L/492L

Review Identification Information

Date of Review:	12/1/23
Project Review is Part of:	Medcurity
Branch/PR Identifier:	PR15 multithreading-scanning
Code Reviewer(s):	Artis Nateephaisan
Code Author or Source of Code:	Colleen Lemak

These following categories should be addressed, though not all might be applicable, it will depend upon the project and code at hand.

#1 Project, Milestone, Issues, Features Included, and/or Pull Request description

This issue addresses some core functionality with the network crawler, primarily scanning for devices and parsing the output to the database.

#2 Code Structural, Design Considerations, and Style Guide Suggestions

The styling and design look good. I like how there are comment blocks before each function in the code that go into what the code does and how it works. Makes it really easy for other members to understand what's going on in reviews.

#3 Code Architectural and Efficiency Considerations

Efficiency seems fine. Architecturally the code is linear and is really easy to read, with core functions that retrieve network info on the top and a main program driver on the bottom.

#4 Execution, Runtime, and Bugs

Runs as expected.

#5 Documentation Quality and Completeness

No documentation so far since there could be potential changes in the crawler further down the line. Overall the functions look good and we should be sticking to the core layout of this crawl-device.py file, so amazing work on that.

#6 Testing, Tests, Coverage, and suggested testing improvements

Tests were run on basic networks (personal networks) but not on client networks yet. Tests have yet to be implemented (will be implemented as we go) into the second semester.

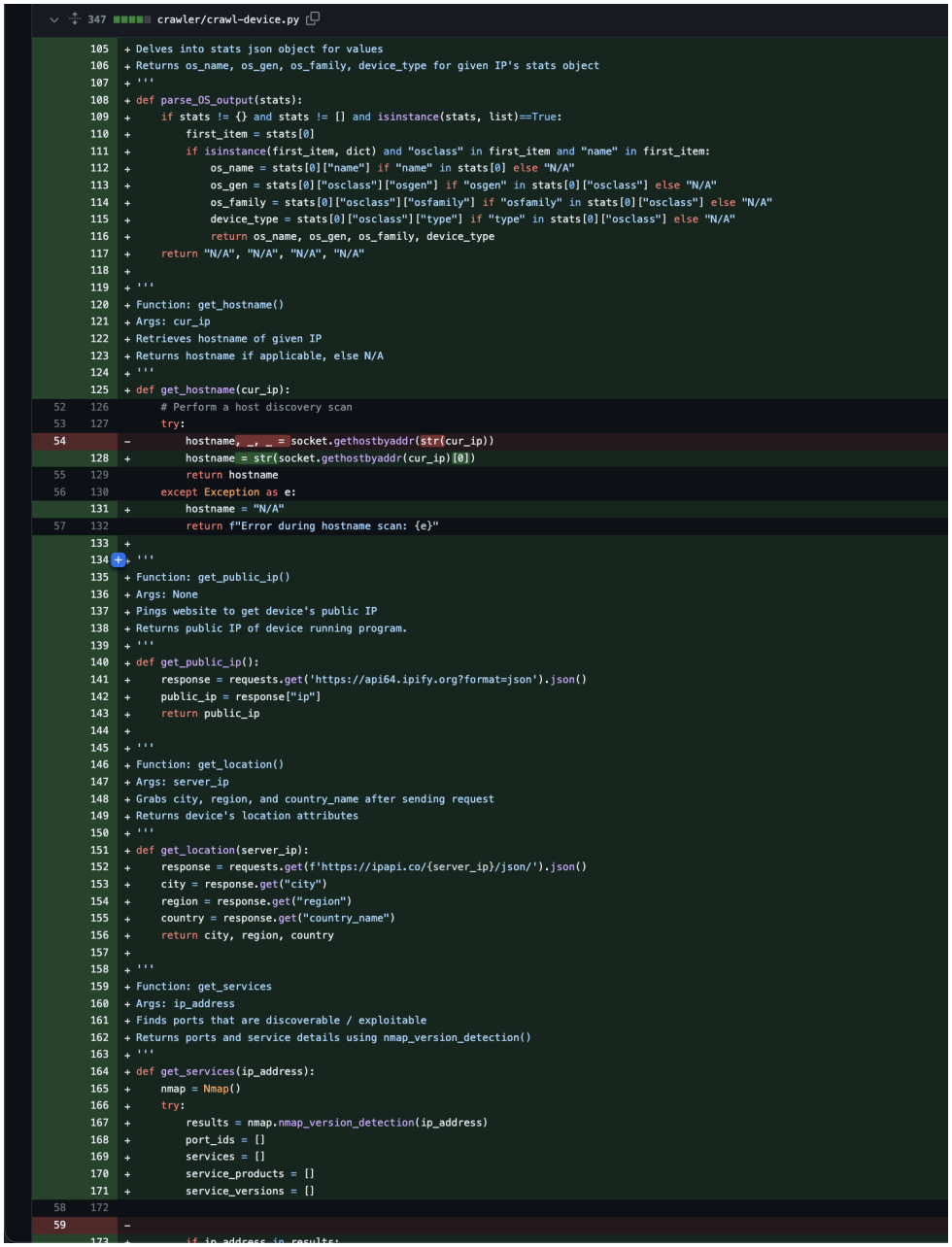
#7 General Comments, Notes or Other Suggestions for Author

I noticed that the author added a ton of comments throughout the code, and honestly it made it really easy to follow along with what the author did. This is something I should add myself to ensure that my group mates can easily follow along with what I have as well. No suggestions other than keep up the helpful comments.

#8 Appendix - Include up to five pages of the code under review as a diff.
See git diff or do a diff in VS code/IDE of choice.
Show what's being reviewed, code or otherwise (config files, scripts, docs, etc).


Changes from all commits
File filter
Conversations
Jump to
347 crawler/crawl-device.py
@@ -4,69 +4,322 @@
from nmap3 import Nmap
import json
import ipaddress
+ import requests
+ import netifaces
import socket
+ from scapy.all import ARP, Ether, srp
+ from pysnmp.hlapi import * # pip install pysnmp
+ import threading
+ from threading import Thread
+ import ssl
- def scan_device(nmap, start_ip, end_ip, dest_filename):
- cur_ip = start_ip
- # Iterate through the list of target IP addresses
- while cur_ip <= end_ip:
- print("\nStarted scanning " + str(cur_ip) + "...")
- # Conduct OS detection scan
- json_results, parsed_obj, stats = get_OS(nmap, cur_ip)
- print("OS Name: " + stats[0]["name"])
- print("OS Gen: " + stats[0]["osclass"]["osgen"])
- print(stats)
- # Write string-results to .json file
- with open(dest_filename,'w') as f:
- f.write(json_results)
- # Crawl for Hostname
- hostname = get_hostname(nmap, cur_ip)
- if not hostname.startswith("Error"):
- print(f"The hostname of the server with IP {cur_ip} is: {hostname}")
+ '''
+ ScannerThread class runs scan_uphosts() and retrieves result with multithreading.
+ '''
+ class ScannerThread(Thread):
+ def __init__(self, argument):
+ Thread.__init__(self)
+ self.argument = argument
+ self.result = None
+ def run(self):
+ self.result = scan_uphosts(self.argument) # store output from function
+ '''
+ ServiceThread class runs get_services() and retrieves result with multithreading.
+ '''
+ class ServiceThread(Thread):
+ def __init__(self, argument):
+ Thread.__init__(self)
+ self.argument = argument
+ self.result = None
+ def run(self):
+ self.result = get_services(self.argument)
+ '''
+ Function: get_default_gateway()
+ Args: None
+ Parses the netifaces.gateways() output to retrieve the default gateway IP.
+ Returns IP address if applicable, else None.
+ '''
+ def get_default_gateway():
+ gateways = netifaces.gateways()
+ if 'default' in gateways and netifaces.AF_INET in gateways['default']: # check attributes exist
+ return gateways['default'][netifaces.AF_INET][0]
+ return None

```
+
+ '''
+ Function: get_network_subnet()
+ Args: gateway_ip
+ Gets CIDR /24 Notation of gateway IP
+ Returns gateway IP's subnet mask.
+ '''
+ def get_network_subnet(gateway_ip):
+     if gateway_ip:
+         gateway_network = ipaddress.ip_interface(f"{gateway_ip}/24")
+         return str(gateway_network.network)
+     return None
+
+ '''
+ Function: get_hosts_up()
+ Args: subnet
+ Constructs and sends packet to get MAC addresses from connected subnet
+ Retuns list of hosts currently up and list of MAC addresses
+ '''
+ def get_hosts_up(subnet):
+     macs_lst = []
+     arp = ARP(pdst=subnet) # use ARP request packet to ping/communicate with hosts
+     ether = Ether(dst="ff:ff:ff:ff:ff:ff")  # cover all IP range values
+     packet = ether/arp
+     result = srp(packet, timeout=3, verbose=False)[0] # send packets out
+     for sent, received in result:
+         mac = received.hwsrc # attribute holds parsed MAC address
+         if mac != "":
+             macs_lst.append(mac)
             else:
-             print(f"Failed to retrieve the hostname. {hostname}")
-
-         cur_ip += 1
-
-     f.close()
-     print("\nProcess finished.\n")
-
-
+             macs_lst.append("N/A")
+     # Record hosts that respond to ARP requests
+     hosts_up = [res[1].psrc for res in result]
+     return hosts_up, macs_lst
+
+ '''
+ Function: get_OS()
+ Args: nmap, cur_ip
+ Requests information from nmap library nmap_os_detection
+ Returns parsed json_results, parsed_obj, and stats objects if applicable, else None
+ '''
   def get_OS(nmap, cur_ip):
       scan_dict = nmap.nmap_os_detection(str(cur_ip))
       json_results = json.dumps(scan_dict, indent=4) # returns type string
-     # Convert string-results into parsed object
       parsed_obj = json.loads(json_results) # returns a json-object
-     try:
-         stats = parsed_obj[str(cur_ip)]["osmatch"]
+     stats = []
+     try: # based on .json output in stats
+         stats = parsed_obj[str(cur_ip)]["osmatch"] if "osmatch" in parsed_obj[str(cur_ip)] else []
+         return json_results, parsed_obj, stats
       except:
           print("No known OS information.")
-     return json_results, parsed_obj, stats
-
-
- def get_hostname(nmap, cur_ip):
+     return None
+
+ '''
+ Function: parse_OS_output()
+ Args: stats
```


347 crawler/crawl-device.py
+ Delves into stats json object for values
+ Returns os_name, os_gen, os_family, device_type for given IP's stats object
+ '''
+ def parse_OS_output(stats):
+ if stats != {} and stats != [] and isinstance(stats, list)==True:
+ first_item = stats[0]
+ if isinstance(first_item, dict) and "osclass" in first_item and "name" in first_item:
+ os_name = stats[0]["name"] if "name" in stats[0] else "N/A"
+ os_gen = stats[0]["osclass"]["osgen"] if "osgen" in stats[0]["osclass"] else "N/A"
+ os_family = stats[0]["osclass"]["osfamily"] if "osfamily" in stats[0]["osclass"] else "N/A"
+ device_type = stats[0]["osclass"]["type"] if "type" in stats[0]["osclass"] else "N/A"
+ return os_name, os_gen, os_family, device_type
+ return "N/A", "N/A", "N/A", "N/A"
+
+ '''
+ Function: get_hostname()
+ Args: cur_ip
+ Retrieves hostname of given IP
+ Returns hostname if applicable, else N/A
+ '''
+ def get_hostname(cur_ip):
# Perform a host discovery scan
try:
- hostname, _, _ = socket.gethostbyaddr(str(cur_ip))
+ hostname = str(socket.gethostbyaddr(cur_ip)[0])
return hostname
except Exception as e:
+ hostname = "N/A"
return f"Error during hostname scan: {e}"
+
+ '''
+ Function: get_public_ip()
+ Args: None
+ Pings website to get device's public IP
+ Returns public IP of device running program.
+ '''
+ def get_public_ip():
+ response = requests.get('https://api64.ipify.org?format=json').json()
+ public_ip = response["ip"]
+ return public_ip
+
+ '''
+ Function: get_location()
+ Args: server_ip
+ Grabs city, region, and country_name after sending request
+ Returns device's location attributes
+ '''
+ def get_location(server_ip):
+ response = requests.get(f'https://ipapi.co/{server_ip}/json/').json()
+ city = response.get("city")
+ region = response.get("region")
+ country = response.get("country_name")
+ return city, region, country
+
+ '''
+ Function: get_services
+ Args: ip_address
+ Finds ports that are discoverable / exploitable
+ Returns ports and service details using nmap_version_detection()
+ '''
+ def get_services(ip_address):
+ nmap = Nmap()
+ try:
+ results = nmap.nmap_version_detection(ip_address)
+ port_ids = []
+ services = []
+ service_products = []
+ service_versions = []
-
+ if ip_address in results:

```
+         if ip_address in results:
+             ports = results[ip_address]['ports']
+             for port in ports:
+                 service = port['service']
+                 port_id = port['portid']
+                 port_ids.append(port_id)
+                 service_name = service.get('name', 'N/A')
+                 services.append(service_name)
+                 service_product = service.get('product', 'N/A')
+                 service_products.append(service_product)
+                 service_version = service.get('version', 'N/A') # grab version if applicable, else N/A
+                 service_versions.append(service_version)
+             return port_ids, services, service_products, service_versions
+         else:
+             print("No information found for the given IP address.")
+     except Exception as e:
+         print(f"Error: {e}")
+     return [], [], [], []
+
+ '''
+ Function: scan_hosts()
+ Args: host address
+ Fetches cur_device_name, os_name, os_gen, os_family, device_type for host using helper functions.
+ '''
+ def scan_uphosts(host):
+     cur_device_name = get_hostname(host) # crawl devices connected to the subnet
+     if cur_device_name.startswith("Error"):
+         cur_device_name = "N/A"
+
+     json_results, parsed_obj, stats = get_OS(nmap, host) # conduct OS detection scan
+     os_name, os_gen, os_family, device_type = parse_OS_output(stats)
+     print(f"\nDevice IP: {host}")
+     print("Hostname: " + cur_device_name)
+     print("Operating System Name: " + os_name)
+     print("Operating System Generation: " + os_gen)
+     print("Operating System Family: " + os_family)
+     print("Device Type: " + device_type)
+     return cur_device_name, os_name, os_gen, os_family, device_type
+
+ '''
+ Function: server_encryption_type()
+ Args: hostname
+ Connect through HTTPS on device or server
+ Return the version of the encryption using ssl if applicable, else Error.
+ '''
+ def get_server_encryption_type(hostname):
+     try:
+         context = ssl.create_default_context()
+         with context.wrap_socket(socket.socket(), server_hostname=hostname) as s:
+             s.connect((hostname, 443))  # Port for HTTPS connections
+             return s.version()
+     except Exception as e:
+         return f"Error: {str(e)}"
+
+ '''
+ Function: fetch_host_stats()
+ Args: up_hosts
+ Create a ScannerThread to execute tasks
+ Returns list of device_names, os_names, os_families, and device_types.
+ '''
+ def fetch_host_stats(up_hosts):
+     # Scan device and software details
+     threads = []; device_names = []; os_names = []; os_gens = []; os_families = []; device_types = []
+     for host in up_hosts:
+         thread = ScannerThread(host)
+         thread.start()
+         print("Started scanning " + host + "...")
+         threads.append(thread)
+     for thread in threads:
+         thread.join()
+     try:
```

```
+     try:
+         for thread in threads:
+             device_name, os_name, os_gen, os_family, device_type = thread.result
+             device_names.append(device_name)
+             os_names.append(os_name)
+             os_gens.append(os_gen)
+             os_families.append(os_family)
+             device_types.append(device_type)
+     except Exception as e:
+         print("Unable to find OS.")
+     return device_names, os_names, os_gens, os_families, device_types
+
+ '''
+ Function: fetch_ports_stats()
+ Args: up_hosts
+ Create a ServiceThread to execute tasks
+ Returns list of port_ids, services, service products, and service versions.
+ '''
+ def fetch_ports_stats(up_hosts):
+     # Port scan and services on device
+     port_ids_lst = []; services_lst = []; service_products_lst = []; service_versions_lst = []
+     threads = []
+     for host in up_hosts:
+         thread = ServiceThread(host)
+         thread.start()
+         threads.append(thread)
+     for thread in threads:
+         thread.join()
+     try:
+         for thread in threads:
+             port_ids, services, service_products, service_versions = thread.result
+             port_ids_lst.append(port_ids)
+             services_lst.append(services)
+             service_products_lst.append(service_products)
+             service_versions_lst.append(service_versions)
+
+             print(f"\nPort Scan on {host}:")
+             for i in range(len(port_ids)):
+                 print(f"\t{i}. Port: {port_ids[i]}\n\t   Service Name: {services[i]}\n\t   Product: {service_products[i]}\n\t   Version: {ser
+     except Exception as e:
+         print("Unable to get discoverable ports")
+     return port_ids_lst, services_lst, service_products_lst, service_versions_lst
+
+ '''
+ Main Program Driver
+ '''
  if __name__ == "__main__":
-     # Network Investigation
-     nmap = Nmap()
+     print("Started Crawler...")
+     nmap = Nmap() # Instantiate nmap object

-     dest_filename = "test_results.json" # "scan_results.json"
-     start_ip = "172.23.96.1"
-     end_ip = "172.23.96.1"
+     public_ip = get_public_ip()
+     city, region, country = get_location(public_ip)
+     gateway_ip = get_default_gateway()
+     server_name = get_hostname(gateway_ip)
+     subnet = get_network_subnet(gateway_ip)
+     up_hosts, macs_lst = get_hosts_up(subnet)
+     num_devices = len(up_hosts)
+     encryption = get_server_encryption_type(gateway_ip)

-     start_ip = ipaddress.IPv4Address(start_ip)
-     end_ip = ipaddress.IPv4Address(end_ip)
+     print(f"\nServer Gateway IP: {gateway_ip}")
+     print(f"Network Subnet: {subnet}")
+     print(f"Number of Connections: {num_devices}")
+     if not server_name.startswith("Error"):
```