Project #4 (Optional): Network exploration and Security auditing

Northwestern CS340, Fall 2022

1 Before starting

- Project #4 should be done in groups of two or individually.
- Please carefully read the instructions including all references and footnotes.

2 Overview

In this project you will write a tool for network exploration and security auditing. Your tool will take as input a list of domains, it will probe those domains/websites in several different ways, and it will print a report detailing some network characteristics and security features/capabilities each domain.

Note: this is a new project, so please let us know if you find any issues with the instructions. If we discover any serious issues that require a significant change in the instructions we will send an announcement via Canvas.

3 Learning goals

After completing this project, students should

- Have a high-level understanding of several important web security protocols/features.
- Know how to
  - Use command-line tools (`OpenSSL`, `nmap`, `telnet`, and `nslookup`) to measure network characteristics.
  - Build a Python tool that make use of command-line utilities.
- Gain experience in
  - Designing software to implement a complex set of interrelated tasks.
  - Choosing from multiple libraries/tools to accomplish a task.
  - Distributing a Python code that depends on many 3rd-party libraries but is still *portable* (someone else can download and run it easily).
- Have a moderately impressive project for your portfolio/resume.

4 Part 1: Scanner Framework

This part is worth 10% of the total grade, if completed correctly.

Write a Python 3 program that takes a list of web domains as an input and outputs a JSON dictionary with information about each domain.

Your program will be invoked as follows:

```
$ python3 scan.py [input_file.txt] [output_file.json]
```

where the parameter is a filename for the input file, which should be in the current directory and should contain a list of domains to test. You can test with the following files (available in the course's public repository[1]), and also write your own input files

1. `test_websites.txt` (Click to download file!)
2. `popular_websites.txt` (Click to download file!)
3. `random_websites.txt` (Click to download file!)

The output of your program will be a JSON[2] dictionary that is output to a file, where its keys are the domains that were scanned and the values are dictionaries with scan results. For example:

```
{
  "northwestern.edu": {
    "scan_time": 1605038710.32,
    "ipv4_addresses": ["129.105.136.48"],
    "ipv6_addresses": [],
    "http_server": "Apache",
    ...
  }
  "google.com": {
    "scan_time": 1605038714.20,
    "ipv4_addresses": ["172.217.6.110", "216.58.192.206", "
       172.217.1.46"],
    "ipv6_addresses": ["2607:f8b0:4009:800::200e"],
    "http_server": "gws",
    ...
  }
}
```

The example above lists four scan results (`scan_time`, `ipv4_addresses`, `ipv6_addresses`, and `http_server`). In Part 2 you will implement the `ipv4_addresses`, `ipv6_addresses`, and `http_server` scanners (and many others), but you should start by just printing `scan_time`, which is the time you started scanning each domain, expressed in UNIX epoch seconds (see this link!).

[1] CS340 Public files repository: https://github.com/northwestern-cs340/fall2022-public-files/tree/main/projects/project-4

[2] Python's JSON library: https://docs.python.org/3/library/json.html

Please make your output human-readable by including indentation. This can be done with a command like:

```
with open(file_name, "w") as f:
json.dump(json_object, f, sort_keys=True, indent=4)
```

There is no provided code for this assignment. Your code may be one file or it may be in multiple files and it may include subdirectories, if needed. You may use any of the standard libraries and you may even `pip install` 3rd-party libraries. However, if you do use 3rd party libraries, please include a `requirements.txt` file that lists the packages that are required. This can be generated by running:

```
$ pip freeze > requirements.txt
```

You will submit a **.tgz** archive of your code, as described later.

Please use a Python virtual environment (it is explained here!) to ensure that you are starting with no 3rd party packages installed. After you create and activate the new virtual environment, you can install additional packages.

5 Part 2: Network Scanners

This part is worth 72% of the total grade, 6% per scanner, if completed correctly.

The subsections below each describe a network scan you must implement. For many of these scans there are command-line tools that can do most of the work. Your Python code can run a command-line process using the subprocess[3] module. For example, try running the following in a Python3 shell:

```
import subprocess
result = subprocess.check_output(["nslookup", "northwestern.edu", "
    8.8.8.8"], timeout=2, stderr=subprocess.STDOUT).decode("utf-8")
print(result)
```

The result string contains the stdout from the command-line invocation of `nslookup northwestern.edu 8.8.8.8`. For example, you can parse this result in your Python code to extract the IPv4 address(es) of northwestern.edu.

In general, you should experiment on the command line with a tool first to find the right syntax before you try to attempt to drive it through Python. Here we give you some recommendations to develop your code

[3] Python's subprocess library: https://docs.python.org/3/library/subprocess.html

- **You should do your development and testing on moore** because the command-line tools on your machine might be slightly different versions, and thus print results in a different format. For example, Windows has an `nslookup` command, but its output is formatted differently than the Linux `nslookup` command on moore. We'll be testing on moore. If you want to make your script robust and platform-independent then you would have to get the work done entirely in Python, without relying on command-line tools (but we're recommending the quick-and-dirty, platform-specific approach).
- **Your code should not crash if a required command-line tool is missing**. For example, Section 5.11 requires the `telnet` command, which is installed on moore but probably not installed on your home machine. If a required command-line tool is missing you must print an error message to **stderr** and skip the particular scan. If a scan is skipped, do not include the corresponding key in the output dictionary.
- **Your code should not hang for a long time if a host is not reachable**. None of the commands you'll run should take more than a few seconds to complete. Notice that in the example code above we included `"timeout=2"` to stop the process after two seconds. You'll have to catch a **TimeoutExpired exception**.

5.1 scan_time

The time when you started scanning the domain, expressed in UNIX epoch seconds (seconds since 1970). The value should be a JSON number (integer or floating point). Output example:

```
"scan_time" : 1605038710.32
```

5.2 ipv4_addresses

A list of IPv4 addresses listed as DNS "A" records for the domain. We suggest you use the `nslookup` command-line tool for this, as demonstrated above. Output example:

```
"ipv4_addresses": ["172.217.6.110", "216.58.192.206", "172.217.1.46"
    ],
```

In more detail

- You can parse the string output of `nslookup` in Python to pick out the IP address answers.
- We want you to print as many IPv4 addresses as you can find.
- There may be multiple answers returned by nslookup (for example, try `nslookup ebay.com`). Include all of them
- Most of these big websites are using some kind of **geographically dynamic DNS**. So, you'll get different answer when you query through resolvers in different locations. Try using all of the DNS resolvers on this list: `public_dns_resolvers.txt`. This list may be hard-coded.

- Many DNS servers will give different results when you query multiple times. For example this query will give different results over time: `nslookup google.com 8.8.8.8`.

5.3 ipv6_addresses

A list of IPv6 addresses listed as DNS "AAAA" records for the domain. Again, we suggest you use the `nslookup` command-line tool for this. You'll have to add `-type=AAAA` to the command. *You may return an empty list if IPv6 is not supported*. All of the details above also apply to IPv6, so we suggest you write a function that runs the same basic scan with different DNS record types (A vs AAAA). Output example:

```
"ipv6_addresses": ["2607:f8b0:4009:800::200e"]
```

5.4 http_server

The web server software reported in the `Server header`[4] of the HTTP response. For this scan (and all the scans below) you may assume that all the IP addresses running the website are configured identically (you may scan just one).

There are many ways to implement this scan, including the `curl` command, Python's `http.client` [5], Python's `requests`[6] library, or an `OpenSSL` connection (as described below).

If no server header is provided in the response, you should set the value to null. Note that this is a special JSON null value, not the string "null" (there are no quotes).

Output examples:

```
# Non-empty response
"http_server" : "Apache/2.2.34 (Amazon)"

# Empty response
"http_server" : null
```

Side note: In Project 1 we used the `telnet` command to create a TCP socket over which we could type out (and debug) the HTTP protocol. If you want to do this with a server that requires an encrypted connection (HTTPS/TLS), you must create an encrypted connection using a command like the following (which connects to "google.com"):

```
$ openssl s_client -crlf -connect google.com:443
```

then to make a simple HTTP request you can type:

[4] server header description https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Server
[5] https://docs.python.org/3/library/http.client.html
[6] https://requests.readthedocs.io/en/master/

```
GET / HTTP/1.0
Host: google.com

<press return twice at the end to create a blank line>
```

You should see the HTTP response headers and some HTML printed to the screen. You may also see some TLS information printed when the encryption parameters are renegotiated.

5.5 insecure_http

Return a JSON boolean indicating whether the website listens for unencrypted HTTP requests on port 80. Output example:

```
"insecure_http": true
```

Notice above that JSON booleans do not have quotes (they are not strings).

5.6 redirect_to_https

Return a JSON boolean indicating whether unencrypted HTTP requests on port 80 are redirected to HTTPS requests on port 443. Note that there several ways for HTTP responses to indicate redirection[7], including:

- 30X response status codes. In this case, check the Location: header to check that it redirects to https.
- HTML meta tags like

```
<meta http-equiv="Refresh" content="0; URL=https://example.com/">
```

  However, this second way (HTML meta tags) is becoming less common and it's not recommended, so you can ignore it.

Note also that you may have to go through a chain of several redirects before being finally redirected to HTTPS. You can give up if the website is broken and redirects you more than 10 times. Return true if you eventually reach an HTTPS page.

If the webserver does not listen to HTTP requests at all, then you should return false. Output example:

```
"redirect_to_https": false
```

[7] https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections

5.7 hsts

Return a JSON boolean indicating whether the website has enabled HTTP Strict Transport Security[8]. This tells the browser to remember to refuse connecting to the domain except with encryption (TLS) enabled. You should check for the appropriate HTTP response header on the final page that you are redirected to. Output example:

```
"hsts": false
```

5.8 tls_versions

List all versions of Transport Layer Security[9] (TLS/SSL) supported by the server, as a list of strings (in no particular order). The options are:

- SSLv2 (obsolete, see https://drownattack.com/)
- SSLv3 (obsolete, see https://disablessl3.com/)
- TLSv1.0
- TLSv1.1
- TLSv1.2
- TLSv1.3

Output example:

```
"tls_versions": ["TLSv1.2", "TLSv1.1", "TLSv1.0"]
```

Note: The powerful network scanning tool `nmap` can provide lots of TLS information with a command like:

```
$ nmap --script ssl-enum-ciphers -p 443 northwestern.edu
```

However, `nmap` does not support the latest version of TLS (1.3), so you should use openssl to test for specific versions of TLS. For example, here is a server that supports TLSv1.3:

```
$ echo | openssl s_client -tls1_3 -connect tls13.cloudflare.com:443
```

and here is one that does not:

[8] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
[9] https://en.wikipedia.org/wiki/Transport_Layer_Security

```
$ echo | openssl s_client -tls1_3 -connect stevetarzia.com:443
```

Note that the version of openssl installed on most Macs does not support TLSv1.3. To test this part you should run on moore. In Python, you can mimic the `echo |` that feeds into the `OpenSSL` command by including `input=b''` in your `check_output` command.

5.9 root_ca

List the root certificate authority (CA) at the base of the chain of trust for validating this server's public key. Just list the "organization name" (you'll find this under "O"). `OpenSSL` can give you this with a command like:

```
$ echo | openssl s_client -connect stevetarzia.com:443
```

If the domain does not support TLS then give null as the value. Output example:

```
"root_ca": "Digital Signature Trust Co."
```

5.10 rdns_names

List the reverse DNS names[10] for the IPv4 addresses listed in "Part 2 - ipv4_addresses" (Section 5.2). You can get these by querying DNS for PTR records. Note that you may get multiple names (or no names) for each IP address; list whatever you get, even if it's an empty list. Output example:

```
"rdns_names": ["ord37s03-in-f110.1e100.net", "ord37s03-in-f14.1e100.
    net",
        "ord30s25-in-f14.1e100.net", "ord30s25-in-f206.1e100.net",
        "ord37s07-in-f14.1e100.net", "ord37s07-in-f46.1e100.net"]
```

SIDE NOTE: you may be wondering what is `1e100.net` ? (The answers is here)

5.11 rtt_range

Print the shortest and longest round trip time (RTT) you observe when contacting all the IPv4 addresses listed in "Part 2 - ipv4_addresses" (Section 5.2). You should list this as a list of two numbers giving the RTT in **milliseconds**: $[min, max]$.

[10] https://en.wikipedia.org/wiki/Reverse_DNS_lookup

We suggest using the following command to measure round trip time. This creates a TCP connection and immediately tears it down. You'll want to record the "real" time it returns:

```
$ sh -c "time echo -e '\x1dclose\x0d' | telnet 172.217.6.110 443"
Trying 172.217.6.110...
Connected to 172.217.6.110.
Escape character is '^]'.

telnet> close
Connection closed.

real    0m0.003s
user    0m0.001s
sys     0m0.001s
```

Above we are using the time command in the bourne shell ("sh"), and the output tells us that the RTT was 3 ms. In your python script you can either run the messy command above or you can implement something like it directly in python using `time.time()`[11] and `socket.socket()`[12]. If you do implement it in python, you should compare your results to the command above to ensure it's consistent.

If the domain is not reachable on any common ports (80, 22, 443), then return a null value. Output example:

```
"rtt_range": [4, 20]
```

5.12 geo_locations

List the set of real-world locations (city, province, country) for all the IPv4 addresses listed in "Part 2 - ipv4_addresses" (Section 5.2). Do not repeat locations. You should use the MaxMind IP Geolocation database via the `maxminddb`[13] Python library. We will provide a recent version of the dataset: `GeoLite2-City_20201103.tar.gz` (Click to Download!)

You should get a file `GeoLite2-City.mmdb` from the archive above and place it in your working directory. The `maxminddb` library requires that database file. When we test your code we will copy the file `GeoLite2-City.mmdb` into the current directory.

You can test your results with this website: https://www.maxmind.com/en/geoip2-precision-demo. Output example:

```
"geo_locations": ["Evanston, Illinois, United States", "New York,
   New York, United States"]
```

[11] https://docs.python.org/3/library/time.html#time.time
[12] https://docs.python.org/3/howto/sockets.html
[13] https://maxminddb.readthedocs.io/en/latest/

6 Part 3: Report

This part is worth 18% of the total grade, if completed correctly.

Write a python script `report.py` that prints an ASCII text report summarizing the results from Part 2. It will take as a parameter a filename for a json file in the format you generated in Part 2. It will print the report to a text file.

Your program will be invoked as follows:

```
$ python3 report.py [input_file.json] [output_file.txt]
```

To make your report readable and attractive, we suggest you use the texttable[14] library. The report should contain:

1. A textual or tabular listing of all the information returned in Part 2, with a section for each domain.
2. A table showing the RTT ranges for all domains, sorted by the minimum RTT (ordered from fastest to slowest).
3. A table showing the number of occurrences for each observed root certificate authority (from "Part 2 - root_ca", Section 5.9), sorted from most popular to least.
4. A table showing the number of occurrences of each web server (from "Part 2 - http_server", Section 5.4), ordered from most popular to least.
5. A table showing the percentage of scanned domains supporting:
   - each version of TLS listed in "Part 2 - tls_versions" (Section 5.8). We expect to see close to zero percent for SSLv2 and SSLv3.
   - "plain http" ("Part 2 - insecure_http", Section 5.5)
   - "https redirect" ("Part 2 - redirect_to_https", Section 5.6)
   - "hsts" ("Part 2 - hsts", Section 5.7)
   - "ipv6" ("Part 2 - ipv6_addresses", Section 5.3)

7 Submission guidelines

- You should work in pairs. List the authors (names and net IDs) in a `README.txt` file.
- Please make just one submission for the pair.
- Your code must compile and run on moore.wot.eecs.northwestern.edu.
- Your submission should be a **.tgz** file (a gzipped tarball), named **NETID1_NETID2.tgz**, including the following files:
  - `README.txt` (listing names and net IDs of the authors).
  - `requirements.txt` (as mentioned above, listing the packages you pip installed)

[14] https://pypi.org/project/texttable/

  - `scan.py`
  - `report.py`
  - any other Python source files you wrote.
  - `scan_out.json` (a copy of the output printed by your `scan.py`, for reference)
  - `report_out.txt` (a copy of the output printed by your `report.py`, for reference)
- Please make sure to include both partners' net IDs in the submission filename.
- Remember that you should leave a comment explaining any bits of code that you copied from the Internet (just in case two groups copy the same code from the Internet, we won't think that you copied each other directly).

8 Testing

Code will be graded on `moore.wot.eecs.northwestern.edu`. Please test your code there before submitting.

After you make your tarball, do the following to make sure it includes everything that's needed for the TA to run it:

- login to moore.
- create a new temporary folder.
- extract your tarball in that folder.
- copy the `test_websites.txt` and `GeoLite2-City.mmdb` files into the folder.
- create a new Python3 virtual environment in that folder:

```
python3 -m venv tmp_env
```

- activate the virtual environment.

```
# If you're using the bash shell:
source tmp_env/bin/activate

# If you're using the tcsh shell:
source tmp_env/bin/activate.csh
```

- pip install the required packages:

```
pip install -r requirements.txt
```

- run the scanner:

```
python scan.py test_websites.txt out.json
```

- run the report

```
python report.py out.json report.txt
```

- view `out.json` and `report.txt` to verify that they look good.