CS 457 – Fall 2022
Project 2: Anonymous Web Get

Language: Python **Total:** 100 Points

Project Description

It is often desirable to retrieve files from the Web anonymously; in other words, performing web transactions without revealing your identity. This is useful if you are located in countries with repressive regimes, if you suspect someone is unlawfully monitoring a server, or if you want to bypass statistics collection on a server. On the Internet, while you don't necessarily need to reveal your true identity, you have to reveal your Internet address, which in turn may be used to identify your computer, which may be linked to you as a person. Note that using DHCP to obtain a different address periodically does not solve the problem, because your ISP typically tracks IP address usage and can link it back to your MAC address (of your computer, router, or modem).

One way to obscure your true identity is to use several computers as **stepping-stones**. The idea is as follows: instead of making your request directly to the server, you send it to a friendly peer. This neighbor, in turn, forwards the request to another peer, who may forward to another, and so on. The chain may be arbitrarily long, depending on latency requirements. The final peer in the chain makes the request to the target server, retrieves the document, and forwards it back along the chain until the file is delivered to the original requestor. As the file is returned, each hop tears down its connection, until the entire chain of connections is destroyed.

Tracking who the original requestor was in this scenario is very hard. One would have to examine state on all machines in the chain, and since such state is only maintained while the connection is active, one has a very small window to follow the chain. Even so, one would either have to log in to every machine or have monitoring equipment in all networks to track the connections. If some of the stepping-stones are located in different countries, this task becomes virtually impossible.

The chain becomes more effective when the pool of stepping-stones is large. Moreover, if the chain is dynamically reconfigured with a very new request, it becomes much harder to be detected by monitoring many requests over time.

Another advantage of stepping-stones is that if the last stone is carefully selected the system can provide access to services that are restricted by source IP address, such as campus resources.

In this project, you will write the necessary code to create a dynamically reconfigurable chain of stepping-stones. Your chain will support a single command - wget - to retrieve specific web documents. You will write TWO modules:

The Reader: The reader is the interface to the stepping stone chain. The reader takes one command line argument, the URL of the file to retrieve. In addition, the reader will read a local *configuration file* that contains the IP addresses and port numbers of all the available steppingstones. Note that having this file on your computer does not compromise the anonymity of the chain: one would still have to prove that your computer was used to retrieve a particular document, in other words having knowledge of the stepping stones does not imply you used them.

The Stepping-Stone (ss): The ss acts both as a server and a client. There is one ss running on each machine. When the SS starts it prints out the hostname and port it is listening to, and then waits for connections. Once a connection arrives, the SS reads the list of remaining ss's in the request and if the list is not empty, strips itself from the list and forwards the request to a randomly chosen ss in the list. If the ss is the last entry on the list, it extracts the URL of the desired document, executes the *os.system()* command to issue a wget to retrieve the document, and then forwards the document back to the previous ss. Once the document is forwarded, the ss erases the local copy of the file and tears down the connection.

Here is a more detailed description of the two modules: **`awget (anonymous wget)`**

You call this `awget.py`. You should do a man wget to see how the standard non-anonymized utility works. awget will have up to two command line arguments:

```
./python awget.py <URL> [-c chainfile]
```

The URL should point to the document you want to retrieve. The chainfile argument is optional and specifies the file containing information about the chain you want to use. The format of the chainfile is as follows:

```
<SSnum>
<SSaddr, SSport>
<SSaddr, SSport>
<SSaddr, SSport>
...
```

where SSnum specifies how many stepping stones are in the file, and the 2-tuples <SSaddr,SSport> specify the address and port each stepping stone is listening on, respectively. Please follow the given sample **chaingang.txt** to make another set of stepping stones.

Since the chainfile argument is optional, if not specified awget should read the chain configuration from a local file called chaingang.txt. If no chainfile is given at the command line and awget fails to locate the chaingang.txt file, awget should print an error message and exit.

Assuming awget reads both parameters (URL and chainfile) correctly, it should next pick a random SS from the file to contact next. One easy way to pick a random SS is to use the python random library. Once awget determines the next SS, it connects to it and sends the URL and the chain list from the file, after it strips the selected entry. awget then waits for the data to arrive, and once it does it saves it into a local file. Note that the name of the file should be the one given in the URL (but not the entire URL). For example, when given URL http://www-net.cs.umass.edu/wireshark-labs/Wireshark_IP_v8.0.pdf the file saved will be named "Wireshark_IP_v8.0.pdf". Also, when URL with no file name is specified, fetch index.html, that is, awget of www.google.com will fetch a file called index.html. Although, you will be tested on URLs that have file names as well as the default index.html.

ss (stepping stone)

You call this ss.py. The ss.py program takes one optional argument, the port it will listen on. ./python ss.py [-p port] Once it starts, ss prints the hostname and port it is running on, and then listens for connections. Once a connection arrives, it reads the URL and the chain information and proceeds as follows. If the chain list is empty: the ss uses the system call os.system() to issue a wget to retrieve the file specified in the URL. Then, it reads the file and transmits it back to the previous ss.py. Once the file is transmitted, the ss tears down the connection, erases the local copy and goes back to listening for more requests. If the chain list is not empty: the `ss` uses a random algorithm similar to `awget` to select the next `ss` from the list. Then, it connects to the next `ss` and sends the URL and the chain list after it removes itself from it. Then, it waits for the file to arrive. Once it does, the `ss` relays that file to the previous `ss`, tears down the connection and goes back to listening for new connections.

Implementation Issues

Your project must be capable of handling multiple concurrent requests, as assignment given in Programming Lab 3: Multiple TCP Connections. . In order to do that, you may use one of the following approaches.

- Make your stepping stones **multithreaded**: this means that when a new connection arrives, a new thread is dispatched to handle it.
- Use a single thread with **select():** this means there is one thread that handles all concurrent requests. To do so you need to use select to manage multiple open sockets at any given time.

How to run your project

First, you will start several instances of `ss` on different machines in the CS lab (We strongly recommend you start with one instance first, for easier debugging). You will then create the chaingang file by noting the hostname and ports the `ss's` are running on. Note that you may ask to get the same port number each time your program starts, which will simplify creating the configuration file. For the final run we suggest you have four ss's.

You then run awget. Choose any file, a pdf, image, or whatever else you like. A document or image will make it easier to determine if your file was received correctly. As the request makes it though the chain, both modules should print clear messages.

Here is an example:

```
awget:
Request: www.cs.colostate.edu/...
chainlist is
<SSaddr, SSport>
<SSaddr, SSport>
<SSaddr, SSport>
```

```
<SSaddr, SSport>
next SS is <SSaddr, SSport>
waiting for file...
..
Received file <filename>
Goodbye!
```

```
ss <SSaddr, SSport>:
Request: www.cs.colostate.edu/...
chainlist is
<SSaddr, SSport>
<SSaddr, SSport>
<SSaddr, SSport>
next SS is <SSaddr, SSport>
waiting for file...
..
Relaying file ...
Goodbye!
```

```
ss <SSaddr, SSport>:
Request: www.cs.colostate.edu/...
chainlist is empty
issuing wget for file <filename>
..
File received
Relaying file ...
Goodbye!
```

Hints/Pointers

Here is a flow chart to help you with implementation.

Chain File Sample

```
4
129.82.45.249 20000
129.82.47.209 25000
129.82.47.223 30000
129.82.47.243 35000
```

Program Flow

ss

1. ss takes one optional argument, the port it will listen on. Example(`./python ss.py -p 20000`)
2. ss prints the hostname and port, it is running on. To find hostname you can use gethostname.
3. Create the socket and fill in its values. Then Bind the socket.
4. Create a loop statement and set the socket in listen mode.
5. Once a connection arrives, it reads the URL and the chain information.
6. Create a thread using pthread_create (or python equivalent) and pass the arguments. [Or use select()]
7. If the chain list is empty:
   a. The ss uses the system call system() (or python equivalent) to issue a wget to retrieve the file specified in the URL.
   b. Reads the file in small chunks and transmits it back to the previous SS. The Previous SS also receives the file in chunks.
   c. Once the file is completely transmitted, the ss should tear down the connection.
   d. Erase the local copy and go back to listening for more requests.
8. If the chain list is not empty:
   a. Uses a random algorithm such as rand() function to select the next SS from the list.
   b. Remove the current ss details from the chain list and send the URL and chainlist to the next ss.
   c. Wait till you receive the fill from the next ss.
   d. Reads the file in small chunks and transmits it back to the previous SS. The Previous SS also receives the file in chunks.
   e. Once the file is completely transmitted, the ss should tear down the connection.
   f. Erase the local copy and go back to listening for more requests.

awget

1. awget will have up to two command line arguments (URL and chainfile)
2. Example `./python awget.py [URL] [-c chainfile]`
3. The URL should point to the document you want to retrieve.
4. Passing the chainfile as an argument is optional. If chainfile is not specified `awget` should read the chain configuration from a local file called “`chaingang.txt`”.
5. If no chainfile is given at the command line and `awget` fails to locate the `chaingang.txt` file, `awget` should print an error message and exit.
6. If `awget` can read both URL and chainfile correctly, proceed.
7. Find a random `ss` from the chainfile list (“`chaingang.txt`”.).
8. Once you have the ss, IP address and port number, create the socket and fill in its values.
9. Send a connect request to the ss.
10. Once the connect request is accepted, strip the ss details from the chainlist and then send the URL and chainlist to the ss.
11. Wait till you receive the file.
12. Create a looping statement to receive the file in chunks.
13. Save the data received in a local file. The file name should be same as the file requested. For Example: For example, when given URL is `https://gaia.cs.umass.edu/kurose_ross/ppt-8e/Chapter_3_v8.0.pptx` the file saved will be named `Chapter_3_v8.0.pptx`
14. When URL with no file name is specified, fetch index.html, that is, awget of www.colostate.edu will fetch a file called `index.html`.

Project 2 Grading Policy

1. The chain of stepping stones is formed correctly - 40 points
2. Last stepping stones retrieves required file correctly - 10 points
3. Stepping stones relay file back to requester correctly - 30 points
4. Stepping stones can handle multiple, concurrent requests - 10 points
5. Outputs according to project description - 10 points

Points

This exercise is worth 100 points. Overall, projects are worth 25% of your total grade. Therefore, this project is worth 15% of your total grade.

Submission Instructions

Turn in your assignment (only **ss.py** and **awget.py**) submitting a tarball (**GroupName.tar/zip**) to CANVAS. Be sure that the files are at the root of the archive and not in any folders. GTAs evaluate the project on lab machines.

Support

For any questions regarding this assignment, email the GTAs with the subject line:
CS457 Fall 2021 Project2 Query at cs457@cs.colostate.edu

You can get supports from the GTAs during their office hours or over email. The following table contains the required contact information.

GTA Name	MD AL AMIN	MICHAEL BOYLE
GTA Office	CSB-120	
GTA Office Hours	T/Th 2:00 – 4:00 pm	Mon/Wed 11:00am-1:00pm
Course Email	cs457@cs.colostate.edu	