CS 457 – Fall 2017

Project 2: Anonymous Web Get

Version 1.0
Date assigned: Saturday, September 23, 2017
Date due: Sunday, October 22, 2017 at midnight
Note: this project counts as 14% of your total semester grade. Grading policy is listed below.

Project Description

It is often desirable to retrieve files from the Web anonymously, in other words 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 (don't forget your cookies!). In 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. The 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 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 stepping stones. 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: 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 `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 will call this executable `awget` (do a `man wget` to see how the standard utility works). `awget` will have up to two command line arguments:

```
  $awget <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.

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 system call `rand()` and then mod N the result, where N is the number of SS' in the file (note that unless you seed `rand()` it will always return the same value, which is great for debugging but not good for anonymity). 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.cs.colostate.edu/~cs457/p2.html` the file saved will be named `p2.html`. 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, I will test you on URLs that have file names.

ss (stepping stone)

You will call this executable ss. ss takes one optional argument, the port it will listen on.

```
$ss [-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 `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`. 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 discussed in class. In order to do that, you may follow 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 (I 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 I suggest you have four `ss`'s.

You will 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!
```

What to turn in

1. makefile: typing "make all" should make both awget and ss. Typing "make awget" or "make ss" should make just the specified module. Typing "make clean" should clean up all object and executable files.
2. awget.h: this file should contain the definition of the data structure for passing the URL and the SS list between awget and SS, and between SS and SS. The data structure should contain the URL, the number of SS entries and the SSaddr and SSport for each entry.
3. awget.c[cc]: the code for awget
4. ss.c[cc]: the code for ss
5. A README file: Here you should put anything you want the TA to know to grade your project. For example, you may indicate if your project works as intended or if there are bugs. If you did not complete your project, tell the TA how far you are, and what does not work.

Note: This project may be performed as an individual assignment or in teams of two people. In the case of teams, you will each be given the identical grade. If there is a complaint that "my partner didn't do much work, I had to do most of it" we will ask each individual to explain the project individually and each person will then be graded individually. So play fair!

REMOTE STUDENTS: if you want to work on a team with someone, use the CANVAS forum to inquire if anyone is interested in working in partnership, and then exchange email to confirm the partnership.

How to submit your project

Submit to CANVAS as a tar file.

Make sure you create a single tar archive with all the files in the root of the archive. Do not archive a directory, only the project files. In other words, when we extract the archive, it should not create a new directory but extract the files in the current directory. Check the man page for tar if you are unsure how to do this.

For any questions regarding this assignment email the TA with subject line:
CS457 Query: Regarding P2

Project 2 Grading Policy

1. All five files submitted (awget.c[cc], awget.h, ss.c[cc], Makefile, Readme) - 10 points
2. Stepping stone compiles - 5 points
3. Stepping stone listens on port specified at command line - 5 points
4. Awget compiles - 5 points
5. The chain of stepping stones is formed correctly - 15 points
6. Last stepping stones retrieves required file correctly - 5 points
7. Stepping stones relay file correctly - 15 points
8. Stepping stones can handle multiple, concurrent requests - 10 points
9. Required file is readable at awget - 20 points
10. Outputs according to project description - 5 points
11. Documentation - 5 points

Hints/Pointers

Here is a flow chart to help you with implementation.

Chain File Sample

```
4
129.82.45.59 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(./ss 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 and pass the arguments.[Or use select()]

7. If the chain list is empty:

    a. The ss uses the system call system() 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.

1. 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:

2. Example ./awget [-c chainfile]
3. The URL should point to the document you want to retrieve.

4. Passing the chainfile as an argument is optional

5. If chainfile is not specified awget should read the chain configuration from a local file called chaingang.txt.

6. 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.

7. If awget can read both URL and chainfile correctly, proceed.

8. Find a random ss from the list. You can use rand() function. Seed the value to get different random number each time.

9. Once you have the ss, IP address and port number, create the socket and fill in its values.

10. Send a connect request to the ss.

11. Once the connect request is accepted, strip the ss details from the chainlist and then send the URL and chainlist to the ss.

12. Wait till you receive the file.

13. Create a looping statement to receive the file in chunks.

14. 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 http://www.cs.colostate.edu/~cs457/p2.html the file saved will be named p2.html

15. 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.