Computer Networks
Fall 2011/12

Lab2 – Content Filtering Proxy Server

Lab1 must be submitted using the course website by **23-1-2015.**
>>> **No late submissions will be accepted.**

Lab 2: Content Filtering Proxy

Abstract:

Content-filtering web proxy server provides control over the content passing through the proxy.
A proxy by its nature performs as a man-in-the-middle to all requests, this way the proxy can inspect the requests and responses of its clients and decide, according to a given policy, if the request/response meets the policy conditions it can continue its original course, or reject the request/response which will lead to canceling of the request/response.

Content-filtering web proxies are being used in commercial companies, non-commercial organizations and even at home. The use of content filtering proxy is to ensure the internet usage is acceptable under the policy the organization is trying to enforce. For example, a company blocks facebook.com during working hours.

Major Milestones:

- Set browser to use proxy.
- Update the web server such that it performs as a proxy. Meaning, the proxy should perform the HTTP request it receives and return the responses.
- Update the web server to enforce policy and keep log of all blocks.
- Logs
- Policy Editor

Milestones in Detail:

1. Set browser to use proxy.

1st step is to set your browser to use a HTTP proxy. Google “set proxy in [browser name]” to get instructions on how to set a proxy in your browser. While setting the proxy, set the IP and port that your proxy server will listen to.

For example, if the proxy server will be on the same machine as the browser and will listen to port 1234, than the proxy server you need to set in the browser is “127.0.0.1” and port 1234.

Once the proxy setting in the browser is all set you will notice that if you try to surf, your browser will reach nowhere. This is because now, instead of surfing to the website you asked, it is automatically sending your request to the proxy server. But because we write the proxy server only in step 2, than in this step (step 1) no one will listen to the browser request.

So let’s continue to the next step and start to write the proxy server itself.

2. Usage of the proxy server.

- The name of the main class is proxyServer.
- The command line format to run the proxy server is:
  ```
  java proxyServer [policy file]
  ```

  For example, if the server uses a policy file called “allow_only_google_policy.ini” than the command line is:
  ```
  java proxyServer allow_only_google_policy.ini
  ```

- For any other command line, the application should trace the usage and quit.

- Update config.ini with “logPath” entry. The entry specify a file that keeps the log of the proxy.
  logPath=”output.log” **[submit with output.log]**

3. Update multithreaded web server to parse HTTP requests from proxy:

When you will perform an HTTP request from the browser it will get to your proxy server.
Notice that when the browser knows it’s sending to HTTP proxy server, it includes in the HTTP request the full path of the requested resource and not just the path to the resource.

For example the following request to www.google.com without HTTP proxy:
```
GET / HTTP/1.1
Host: www.google.com
```

Would turn to:
```
GET http://www.google.com/ HTTP/1.1
Host: www.google.com
```

For each request that comes from browser the web server should perform the HTTP request for the browser, receive the response and return it to the browser.

At this point the browser might request on the same connection more embedded objects (persistent HTTP, remember?) than once the thread returns the page to the browser it should try and read more data from the socket for more requests. The socket will close when the client initiates closing.

Notice that params_info.html must be removed from your server.

4. Update the web server to enforce policy and keep log of all blocks.

Policy File:
Read the policy file (2nd command line argument) and enforce the policy written in the file.

There are 4 possible policy rules:

1. block-site "[keyword]" – block HTTP request if the **host** contains the *keyword*.
2. block-resource "[keyword]" – block HTTP request if the requested **resource** contains the *keyword*.
3. block-ip-mask "[IP]/[mask]" – example: 10.15.24.5/24

The policy file format is as follows:
[policy rule 1][CRLF]
[policy rule 2][CRLF]
.
.
.

Example for a policy rule:
block-site "google.com/plus"
block-site "google.com/mail"
block-resource ".avi"

According to the policy file, "google.com/calendar", "facebook.com" is allowed.

In case of a blocked resource, the proxy returns 403 HTTP response code (access denied). In the body of the response you return to the browser the rule that blocked his request.

5. Logs:

Each block that occurs, your proxy server keeps in a log in the logPath file. If the file doesn't exist, your application must create one.
The information kept for each block is:

- Time of block.
- HTTP request that has been blocked.
- Rule blocked the request.

When the browser surfs to the URL http://content-proxy/logs, than the proxy returns an HTML page with the logs written in the file. If the log is empty, return to the user a message that there were no blocks.

6. Policy Editor:

When the browser surfs to the URL http://content-proxy/policies, than the proxy returns an HTML page with the policy file. The HTML page allows the user to add/delete/modify each policy. Every time a policy is being added/deleted/modifies the proxy server reloads the policy file to support the change.

Exception Handling:

The program must not crash!

Use exception handling to recover. If you cannot recover, print what happened to the console and close the application gracefully.

This includes parsing the config.ini, policy.ini or anything being done during the initialization of the application.

Notice that crashing will drop **many** points!

Traces (a.k.a. print to console):

The following traces are **required (on top of those in lab 1)**:

- Every block entry being written to the log file.

You are **encouraged** to use additional traces in your code!
Traces will help you debug your application and perform a better QA.
Make sure your traces actually mean something that will help to understand better what is going on during runtime. **That can and will save you a lot of time!**

Bonuses:

Feel free to add more functionality to the application as you see fit!
Good ideas will be **rewarded** with points and world-wide fame (score may be over a 100)!!!

Please write and explain all the implemented bonuses in a file named bonus.txt.
Notice that bonuses that will not be mentioned in bonus.txt will not be checked, hence they will be ignored!

Important: We don't guarantee in advance that for extra-functionality, extra-points will be rewarded. Also, the idea is to ***add*** a new functionality ***not replace a requested functionality*** – that will lead to dropping of points! In other words, do not ignore things we require, and implement other things instead and call it a bonus! We cannot make it any clearer than that!

Discussion Board:

Please, use the forum of the course to ask questions when something is unclear.

You must make sure that you check the discussion board. If there are unclear matters about the lab that were raised in the discussion board, **our answers are mandatory**, and apply to **everyone**!

You can register to receive e-mail notification from the forum.

A word from your grader:

- **Do not** create your own packages, all your classes **must be in the same package**, and compile while all sources in the same directory.
- If you are coding using a mac, make sure you remove all hidden directories generated by your IDE.
- You can implement the lab using up to JDK1.8 .

Remember - A happy grader is a merciful checker!

Submission:

- You **must** submit a batch file called “compile.bat” - The batch file compiles your code successfully. If the batch fails to compile, **many** points will be dropped.
- You **must** submit a batch file called “run.bat” - The batch file starts your server. If the batch fails to start the server, **many** points will be dropped.

Submit the lab following the course submission rules.
The submitted zip must contain:

- Sources (*.java)
- config.ini
- compile.bat
- run.bat
- Directory called “serverroot” containing your non-code files (your HTMLs + extra files/images etc.)
- bonus.txt with the bonuses you implemented – if applicable.
- **readme.txt** that explains what exactly you have implemented and the role of each class in your server.

Important: we compile and check your project without any IDE (using the batches alone), make sure your application works correctly when executed without an IDE.

Notice: not submitting one of the files needed requested (exclude bonus.txt) will lead to dropping of points!

That’s it ☺ !
Good Luck!