University of Dublin,
Trinity College

Computer Networks (CS3D3)

Project 2 - Proxy Server

Author:
Edmond O'Flynn 12304742

Lecturer:
Prof. Hitesh Tewari

April 5, 2016

Contents

1 Source Code

1.1 Blacklist.java

```
package com.syzible.proxyserver;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

public class Blacklist {
   /**
    * Returns a block message to the client for trying to access
    * a blocked website and handles the socket connections
    * @param url url to be checked for being on the blacklist
    * @param clientSocket socket accepted from the client
    */
   public static void blockWebsite(String url, Socket clientSocket) {
      try {
         System.out.println(url + " is a blacklisted website!");
         String message = "<html>"
               + "<head>"
               + "<title>Blocked!</title>"
               + "<meta charset=\"utf-8\">"
               + "</head>"
               + "<body>"
               + "<h1><font face=\"verdana\">WARNING!</font></h1>"
               + "<p><font face=\"verdana\" color=\"red\" size=\"4\">" + url + "</font>"
               + "<font face=\"verdana\" size=\"4\"> has been blacklisted from use.</font></p>"
               + "<p><font face=\"verdana\" size=\"4\">Please contact your system
                  administrator.</font></p>"
               + "</body>"
               + "</html>\r\n";

         byte[] message_b = message.getBytes();
         final OutputStream to_c = clientSocket.getOutputStream();
         final InputStream stream = new ByteArrayInputStream(
               message.getBytes(StandardCharsets.UTF_8));

         int bytesRead2;
         try {
            while ((bytesRead2 = stream.read(message_b)) != -1) {
               to_c.write(message_b, 0, bytesRead2);
               to_c.flush();
            }
         } catch (Exception e) {
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
```

```
   /**
    * Checks the url provided if it exists on the blacklist
    * @param url url to be checked
    * @return boolean true or false for if the url is on the list
    */
   public static boolean checkList(String url) {
      try {
          String currentLine = "";
         File file = new File("./admin/Blacklist.txt");
         Scanner scanner = new Scanner(file);
         while(scanner.hasNext()) {
            currentLine = scanner.next();
            if(currentLine.contains(url)) {
               System.out.println(url + " is a blacklisted website!");
               scanner.close();
               return true;
            }
         }
         scanner.close();
      } catch(Exception e) {
         e.printStackTrace();
      }
      System.out.println(url + " is a whitelisted website!");
      return false;
   }

}
```

1.2 Caching.java

```
package com.syzible.proxyserver;
import java.io.File;
import java.io.FileOutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class Caching {

   public static final String BASE_DIR = "./admin/Cache/";
   public static final String BASE_DIR_LOG = "./admin/Log.txt";

   /**
    * Stores a record of websites requested with the current time and date
    * @param url website requested by the user
    */
   public static void logHistory(String url) {
      try {
         String timeStamp = new SimpleDateFormat("HH:mm:ss
             dd/MM/yyyy").format(Calendar.getInstance().getTime());
         String log = url + " accessed at " + timeStamp + "\n";
         Path filePath = Paths.get(BASE_DIR_LOG);
```

```
            Files.write(filePath, log.getBytes(), StandardOpenOption.APPEND);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Checks whether or not the cache is still valid for the given proxied website store
     * @param cacheVersion hash key associated with longevity of cache
     * @param uri uri to be parsed for folder acquisition
     * @return boolean for update
     */
    public static boolean checkCached(String cacheVersion, String uri) {
        File directory = new File(BASE_DIR + uri);
        if(!directory.exists()) {
            boolean update = false;
            try {
                directory.lastModified();
                if(!cacheVersion.equals(directory.getName())) {
                    update = true;
                }
            } catch(Exception e) {
                e.printStackTrace();
            } finally {
                if(update) {
                    System.out.println("Updating cache for folder (" + uri + ")");
                    return true;
                }
            }
        }
        return false;
    }

    /**
     * Creates a directory pertaining to the website requested
     * @param uri host being pinged
     */
    public static void createDir(String uri) {
        File directory = new File(BASE_DIR + uri);
        if(!directory.exists()) {
            boolean result = false;
            try {
                directory.mkdir();
                result = true;
            } catch(Exception e) {
                e.printStackTrace();
            } finally {
                if(result) {
                    System.out.println("Caching folder for " + uri + " has been created!");
                }
            }
        }
    }

    /**
```

```
    * Stores and recursively downloads the appropriate file to be cached
    * @param contents data being cached
    * @param uri host name pinged by the request
    */
   public static void saveFile(String contents, String uri) {
       String extension = uri.substring(uri.lastIndexOf("."));
      if(!extension.contains(".com") || !extension.contains(".ie")){
         try {
            String fileDir = BASE_DIR + uri + "/";
            System.out.println(fileDir);

            byte data[] = contents.getBytes();

            File file = new File(fileDir);
            if(file.exists()) {
               FileOutputStream out = new FileOutputStream(fileDir);
               out.write(data);
               out.close();
            } else {
               createDir(uri);
               file.createNewFile();
               FileOutputStream out = new FileOutputStream(fileDir);
               out.write(data);
               out.close();
            }

            System.out.println("File cached successful in " + fileDir);
         } catch(Exception e) {
            e.printStackTrace();
         }
      }
   }
}
```

1.3 Helpers.java

```
package com.syzible.proxyserver;
import java.util.regex.Pattern;

public class Helpers {
   public static final int PORT = 2016;
   public static final int NON_CONNECT_PORT = 80;
   public static final int CONNECT_PORT = 443;

   public static final String HOST = "localhost";
   public static final String USER_AGENT = "Mozilla/5.0";

    public static final Pattern GET_PATTERN =
         Pattern.compile("GET http://(.*)/ HTTP/(1\\.[01])",
            Pattern.CASE_INSENSITIVE);

   public static final Pattern CONNECT_PATTERN =
         Pattern.compile("CONNECT (.+):(.+) HTTP/(1\\.[01])",
         Pattern.CASE_INSENSITIVE);
}
```

1.4 Pooling.java

```
package com.syzible.proxyserver;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.URI;

public class Pooling extends Thread {
   private Socket clientSocket;

   public Pooling(Socket clientSocket) {
      this.clientSocket = clientSocket;
   }

   /**
    * Thread handler for filtering and conduit behaviour for connections
    * as a middle-man for connection forwarding of SSL and HTTP
    */
   @Override
   public void run() {
      outside: try {
         BufferedReader fromClient = new BufferedReader(
               new InputStreamReader(clientSocket.getInputStream()));
         DataOutputStream toClient = new DataOutputStream(
               clientSocket.getOutputStream());

         Socket server = new Socket();

         String host = null;
         int port = -1;

         // Read the request line
         String line = fromClient.readLine();
         String firstLine = line;

         if (line == null) {
            break outside;
         }

         // Prints first line
         String[] tokens = line.split(" ");
         System.out.println(">>> " + tokens[0] + " " + tokens[1]);

         // Extract host and port
         StringBuffer part = new StringBuffer();
         boolean foundHost = false;
         while (!foundHost && line != null && !line.equals("")) {
            if (line.contains("keep-alive")) {
               line = line.replaceAll("keep-alive", "close");
```

```
        }
        if (line.contains("HTTP/1.1")) {
            line = line.replaceAll("HTTP/1.1", "HTTP/1.0");
        }

        if (line.toLowerCase().startsWith("host")) {
            foundHost = true;
            String[] hostTokens = line.split(":");
            host = hostTokens[1].trim();
            if (hostTokens.length > 2) {
                port = Integer.parseInt(hostTokens[2]);
            } else {
                URI uri = new URI(firstLine.split(" ")[1]);
                port = uri.getPort();
            }
        }

        if (line.toLowerCase().startsWith("get")) {
            line = line.replaceFirst("http://", "");
            String rep = line.split(" ")[1]
                  .substring(line.split(" ")[1].indexOf("/"));
            line = line.replaceAll(line.split(" ")[1], rep);
        }

        part.append(line + "\n");
        line = fromClient.readLine();
    }

    // HTTP CONNECT Tunneling
    if (tokens[0].equals("CONNECT")) {
        if (port == -1) {
            port = Helpers.CONNECT_PORT;
        }

        // Connect to server
        try {
            server.connect(new InetSocketAddress(host, port));
        } catch (Exception e) {
            // Fails to connect to server
            toClient.write("HTTP/1.0 502 Bad Gateway".getBytes());
            toClient.write("\r\n\r\n".getBytes());
            e.printStackTrace();
        }

        // Successfully connected to server
        try {
            toClient.write("HTTP/1.0 200 OK".getBytes());
            toClient.write("\r\n\r\n".getBytes());
        } catch (Exception e) {
        }

        final byte[] request = new byte[4096];
        byte[] response = new byte[4096];

        final InputStream from_c = clientSocket.getInputStream();
```

```
      final OutputStream to_c = clientSocket.getOutputStream();

      final InputStream from_s = server.getInputStream();
      final OutputStream to_s = server.getOutputStream();

      Thread clientThread = new Thread() {
         public void run() {
            int bytesRead;
            try {
               while ((bytesRead = from_c.read(request)) != -1) {
                  to_s.write(request, 0, bytesRead);
                  to_s.flush();
               }
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      };
      clientThread.start();

      int bytesRead2;
      try {
         while ((bytesRead2 = from_s.read(response)) != -1) {
            to_c.write(response, 0, bytesRead2);
            to_c.flush();
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      server.close();
      clientSocket.close();
      break outside;
   }

   //
       ------------------------------------------------------------------------------
   // Non-CONNECT HTTP requests
   // look at content length header

   if (port == -1) {
      port = Helpers.NON_CONNECT_PORT;
   }

   if (Blacklist.checkList(host)) {
      Blacklist.blockWebsite(host, clientSocket);
      server.close();
      clientSocket.close();
      break outside;
   } else {
      // Forward request
      server.connect(new InetSocketAddress(host, port));
      PrintWriter toServer = new PrintWriter(
            server.getOutputStream(), true);
      toServer.print(part.toString());
```

```
            // Write the rest of headers
            while (line != null && !line.equals("")) {
               if (line.contains("keep-alive")) {
                  line = line.replaceAll("keep-alive", "close");
               }
               if (line.contains("HTTP/1.1")) {
                  line = line.replaceAll("HTTP/1.1", "HTTP/1.0");
               }

               toServer.println(line);
               line = fromClient.readLine();
            }

            toServer.println("\r\n\r\n");

            // Get response from server
            // Forward to client
            InputStream fromServer = server.getInputStream();
            int bytesRead = 0;
            byte[] response = new byte[4096];
            while ((bytesRead = fromServer.read(response)) != -1) {
               toClient.write(response, 0, bytesRead);
               toClient.flush();
            }
            Caching.logHistory(host);
            Caching.saveFile(response.toString(), host);
            server.close();
            clientSocket.close();
         }

      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

1.5 Proxy.java

```
package com.syzible.proxyserver;
import java.net.ServerSocket;
import java.net.Socket;

public class Proxy extends Thread {

   public Proxy() {
      super("Proxy Server");
   }

   public static void main(String[] args) {
      (new Proxy()).run();
   }

   /**
    * Spawns threads for connection and superuser handling
    */
```

```
    @Override
    public void run() {
        new SU().start();

        try(ServerSocket proxySocket = new ServerSocket(Helpers.PORT)) {
            Socket socket;
            System.out.println("Proxy server started on " + Helpers.HOST + ":" + Helpers.PORT);
            try {
                while((socket = proxySocket.accept()) != null) {
                    new Pooling(socket).start();
                }
            } catch(Exception e) {
                e.printStackTrace();
            }
        } catch(Exception e) {
            e.printStackTrace();
            return;
        }
    }
}
```

1.6 SU.java

```
package com.syzible.proxyserver;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.Scanner;

public class SU extends Thread {

    private static final String BLACKLIST_PATH = "./admin/Blacklist.txt";

    public SU() {
        suPrint("Superuser module invoked.");
        suPrint("Type su --<cmd> to invoke an administrator command.");
    }

    /**
     * Background task handling for receiving input as an admin
     */
    @Override
    public void run() {
        try {
            String input;
            String[] parts;
            Scanner scannerInput = new Scanner(System.in);
            while(scannerInput.hasNext()) {
                input = scannerInput.nextLine();
                parts = input.split(" ");
                if(parts[0].equals("su")) {
                    if(parts[1].equals("--help")) {
```

```
                suPrint("\nHelp invoked!");
                suPrint("Commands for Superuser mode:");
                suPrint("--help -> invokes list of commands");
                suPrint("--block <website> -> adds website to blacklist");
                suPrint("--list -> lists the currently blacklisted websites");
                suPrint("--unblock <website> -> removes website from blacklist");
            } else if(parts[1].equals("--block")) {
                if(!parts[2].equals(null)) {
                    addToBlacklist(parts[2]);
                    suPrint(parts[2] + " has been added to the blacklist!");
                } else {
                    suPrint("Website cannot be null!");
                }
            } else if(parts[1].equals("--list")){
                suPrint("Blacklisted websites:");
                for(String website : blacklistedWebsites()) {
                    suPrint(website);
                }
            } else if(parts[1].equals("--unblock")) {
                if(!parts[2].equals(null)) {
                    suPrint(parts[2] + " has been removed from the blacklist!");
                } else {
                    suPrint("Website cannot be null!");
                }
            } else {
                suPrint("Commands must be prefixed with \"--\"!");
            }
        } else {
            suPrint("Must append \"su\" to the start of the command! su <cmd>
                <parameters>");
        }
      }
      scannerInput.close();
   } catch(Exception e) {
      e.printStackTrace();
   }
}

/**
 * Adds the url as a parameter to the blacklist
 * @param url url to be blocked
 */
public void addToBlacklist(String url) {
   try {
      Path filePath = Paths.get(BLACKLIST_PATH);
      Files.write(filePath, url.getBytes(), StandardOpenOption.APPEND);
   } catch(Exception e) {
      e.printStackTrace();
   }
}

/**
 * Retrieves the list of websites for comparison
 * @return list of blacklisted websites
 */
```

```
   public String[] blacklistedWebsites() {
      ArrayList<String> blacklist = new ArrayList<String>();

      try {
         File file = new File(BLACKLIST_PATH);
         Scanner scanner = new Scanner(file);
         while(scanner.hasNext()) {
            blacklist.add(scanner.next());
         }
         scanner.close();
      } catch(Exception e) {
         e.printStackTrace();
      }

      String[] blacklistedWebsites = new String[blacklist.size()];
      for(int i=0; i < blacklist.size(); i++) {
         blacklistedWebsites[i] = blacklist.get(i);
      }

      return blacklistedWebsites;
   }

   /**
    * Prints a phrase provided to be returned by the superuser
    * @param phrase phrase to be reported by SU
    */
   public static void suPrint(String phrase) {
      System.out.println("SU: " + phrase);
   }

}
```