Activity VIII: Secure Software/Simple Web Server

Instructor : Krerk Piromsopa, Ph.D

Some problems and the examples are taken/adopt from Google Web Security Programming Problem Set[1]. Some parts have been modified to serve the purpose.

Overview

We will use JAVA in this example. Technically, you may do it in your favorite programming languages. We will learn the importance of Secure by Design by designing a simple web server.

The provided SimpleWebServer is a single thread/session web server. While serving a purpose, it is far from being secure. A connection to the server will block others from accessing and may result in **denial of service**. In addition, there is no access control at all. This makes it vulnerable to **elevation of privileges**, **information disclosure** and **tampering**. There is also no log for non-**repudiation.**

You are now in charge of securing it. Please apply the knowledge of secure by design to secure this SimpleWebServer. Keep in mind that this is an educational assignment. Don't try to use it in any real production. It takes more effort than you can imagine to secure a software.

Exercises

1) Here is the source code for SimpleWebServer. Compile and run this program using the following command:

```
C:\SimpleWebServer> javac SimpleWebServer.java
C:\SimpleWebServer> java com.learnsecurity.SimpleWebServer
```

Make sure you have an"index.html" file in your SimpleWebServer directory. To access the server, start up a web browser and enter the following url:

```
http://localhost:8080/index.html
```

```
/***********************************************************************

  SimpleWebServer.java


  This toy web server is used to illustrate security vulnerabilities.
  This web server only supports extremely simple HTTP GET requests.
```

[1] https://github.com/DanielMSchmidt/it-sec/blob/master/serie5/SimpleWebServer.java

```
   This file is also available at http://www.learnsecurity.com/ntk

*****************************************************************/

package com.learnsecurity;

import java.io.*;
import java.net.*;
import java.util.*;

public class SimpleWebServer {

    /* Run the HTTP server on this TCP port. */
    private static final int PORT = 8080;

    /* The socket used to process incoming connections
       from web clients */
    private static ServerSocket dServerSocket;

    public SimpleWebServer () throws Exception {
       dServerSocket = new ServerSocket (PORT);
    }

    public void run() throws Exception {
       while (true) {
           /* wait for a connection from a client */
           Socket s = dServerSocket.accept();

           /* then process the client's request */
           processRequest(s);
       }
    }

    /* Reads the HTTP request from the client, and
       responds with the file the user requested or
       a HTTP error code. */
    public void processRequest(Socket s) throws Exception {
       /* used to read data from the client */
       BufferedReader br =
           new BufferedReader (
                           new InputStreamReader (s.getInputStream()));

       /* used to write data to the client */
       OutputStreamWriter osw =
           new OutputStreamWriter (s.getOutputStream());

       /* read the HTTP request from the client */
       String request = br.readLine();

       String command = null;
       String pathname = null;

       /* parse the HTTP request */
       StringTokenizer st =
           new StringTokenizer (request, " ");

       command = st.nextToken();
       pathname = st.nextToken();

       if (command.equals("GET")) {
           /* if the request is a GET
              try to respond with the file
              the user is requesting */
           serveFile (osw,pathname);
       }
```

```
        else {
            /* if the request is a NOT a GET,
               return an error saying this server
               does not implement the requested command */
            osw.write ("HTTP/1.0 501 Not Implemented\n\n");
        }

        /* close the connection to the client */
        osw.close();
    }

    public void serveFile (OutputStreamWriter osw,
                          String pathname) throws Exception {
        FileReader fr=null;
        int c=-1;
        StringBuffer sb = new StringBuffer();

        /* remove the initial slash at the beginning
           of the pathname in the request */
        if (pathname.charAt(0)=='/')
            pathname=pathname.substring(1);

        /* if there was no filename specified by the
           client, serve the "index.html" file */
        if (pathname.equals(""))
            pathname="index.html";

        /* try to open file specified by pathname */
        try {
            fr = new FileReader (pathname);
            c = fr.read();
        }
        catch (Exception e) {
            /* if the file is not found,return the
               appropriate HTTP response code  */
            osw.write ("HTTP/1.0 404 Not Found\n\n");
            return;
        }

        /* if the requested file can be successfully opened
           and read, then return an OK response code and
           send the contents of the file */
        osw.write ("HTTP/1.0 200 OK\n\n");
        while (c != -1) {
            sb.append((char)c);
            c = fr.read();
        }
        osw.write (sb.toString());
    }

    /* This method is called when the program is run from
       the command line. */
    public static void main (String argv[]) throws Exception {

        /* Create a SimpleWebServer object, and run it */
        SimpleWebServer sws = new SimpleWebServer();
        sws.run();
    }
}
```

a) What happens if a client connects to SimpleWebServer, but never sends any data and never disconnects? Test this out with your running server. What type of an attack is this? (You may use “telnet localhost 8080” and leave it on.)

b) Try the DoS attack described in class: See if you can download /dev/random (assuming you are running on a Linux system). Rewrite serveFile() as discussed in class to guard against this type of attack.

c) Implement logging in SimpleWebServer. For each client that connects to the server, obtain information about that client and write the information in a log file.

d) HTTP supports a mechanism that allows users to upload files in addition to retrieving them through a PUT command.
   - What threats would you need to consider if SimpleWebServer also provided functionality for uploading files?
   - For each of the specific threats you listed, what security mechanisms must be added to mitigate these threats?
   - Implement uploading capability in SimpleWebServer. You will need to research how to send an HTTP PUT command to the server. You also need to write a storeFile() function. Implement as much security as you feel is needed to guard against the threats you specified above.
   - Once you have logging and storeFile() implemented, launch an attack to *deface* index.html, that is, replace it with another index.html that you have created. (This is a common attack against www.whitehouse.gov.)

2) What are the most important steps you would recommend for securing a new web server? A new web application?
3) What is "Cross-Site Scripting"? What is the potential security impact to servers and clients?

4) What are phishing and pharming? What are the ways to protect against such attacks?

5) Explore the OWASP website

   - What are some of the vulnerabilities of web browsers?
   - What are some modes of attack used to implement a Denial of Server? What preventive measures can be implemented?

Some of the problems above, and the SimpleWebServer source code are from:

- "Foundations of Security: What Every Programmer Needs To Know" (ISBN 1590597842) by Neil Daswani, Christoph Kern, and Anita Kesavan.