HTML OBFUSCATION

By

Capt Abdul Rehman Raza Khan (Group Leader)

Capt Shorahbeel Bin Zahur

NC Danyal Sajid

Supervisor:

Dr. Hammad Afzal

Submitted to the Faculty of Computer Science

National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of a B.E Degree in Computer Software Engineering

June 2013

CERTIFICATE

Certified that the contents and form of project report entitled **“HTML Obfuscation”** submitted by 1) Capt Abdul Rehman Raza, 2)Capt Shorahbeel Bin Zahur, and 3) NC Danyal Sajid have been found satisfactory for the requirement of the degree.

Supervisor: _____

Dr. Hammad Afzal

ABSTRACT

The aim of HTML obfuscation was to develop an application that could be installed over a web browser and help obfuscate information that is sent across the internet. Auto-bots and web-crawlers are used to obtain the information, by scrapping, in a mass manner. This information then is used for advertisement, spam, and other malicious purposes. Obfuscation is necessary to break patterns in the source code so as to render the auto-bots ineffective for scraping data from web pages. The application allows the web-server administrator to configure the application for multiple web sites (hosted on the server) and select text for obfuscation via input of patterns.

The application listens to all the web requests (over port 80) and calls for the requested source code from the IIS Server. It then parses the complete code and recognizes patterns that are to be obfuscated. After obfuscation, the new equivalent code is sent back over the web. Obfuscated code is randomly created every time. The basic aim of this application is to multiply the effort of bot-creation and cause de-motivation.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

To Our Parents and Wives for their prayers and support

and

To Google, which made it all possible

ACKNOWLEDGMENTS

We are grateful to our parents, families for their support and prayers. Their faith kept us going.

We are extremely grateful to our project supervisor Dr. Hammad Afzal for his support and guidance without which we could not have moved on with the research. We are thankful to our teachers and instructors here at Computer science department, MCS, all of them have guided us have made it possible for us to complete both this degree and the project.

Table of Contents

1. Introduction

a. Background

Hypertext Markup Language (HTML) is widely used to display content in a web browser. The data in HTML requests is very easily accessible and can be extracted using automated tools (using the source code). This information is used in mass communication, advertisement, intellectual property theft etc.

Obfuscation is the hiding of intended meaning in communication by making communication confusing, willfully ambiguous, and harder to interpret.

b. Problem Statement

Automated tools like web crawlers and auto-bots are used to scrape mass amount of data from the web pages. The target is to gather large chunks of data that can be used for either malicious purposes or mass advertisements. This is usually done be obtaining personal information such as email addresses, phone numbers facebook ids etc. They can spam using this information, cause identity theft, information theft (like articles, research papers etc).

Keeping in mind the above and the fact that textual information is very sensitive and can cause damage to reputation, the team is working on obfuscating this information on the go. This application shall be installed on a web server and provide obfuscation at runtime.

c. Objectives

The objective is to develop an application that shall be installed on a web server. It shall allow the administrator to choose what content, of a web site, to obfuscate while sending the reply over internet.

d. Deliverables

1st Progress Report: including SRS Document

2nd Progress Report: including System Design

3rd Progress Report: including Interface Design

4th Progress Report: including Demonstration

Final Report: including complete documentation

e. Technological Requirements

A Web Server (or PC Configured as such)

Windows OS (XP/Vista/7)

Microsoft IIS

Microsoft Visual Studio 2008

512 Mb Ram (minimum), although it depends on traffic at the web server.

2. Literature Review

a. Previous Work

Obfuscation for software, and static obfuscation tools already exist over the web. These static tools ask the user to provide them with the source code and return the obfuscated equivalent code that can be used to host the site. There have been multiple research papers on software obfuscation (scrambling the code of a software such as its executable's), but not much on HTML obfuscation. The techniques used till now are very random and no fix technique to scramble the code exist (everyone uses its own). Although the generation of this scrambled code is done by replacement of java script (this remains one constant throughout).

b. Shortcomings

The Existing tools do not provide runtime obfuscation and do not provide any kind of flexibility to what kind of information to be scrambled and which to be left as such (in original form).

c. Issues solved by this "HTML OBFUSCATION"

Developing a static HTML obfuscator that generates an equivalent random/scrambled code will not solve the problem completely. Even after the obfuscation, patterns can be detected and code can be broken if it does not change every time the request is made. Therefore, the team has developed an application that can be installed on the web server where the web sites are hosted.

Our application has catered for the following:

- Provides runtime obfuscation of the html code
- Obfuscation is done with each request randomly
- Administrator can choose what to obfuscate and what not to.
- Administrator can provide obfuscation for multiple web sites at once.

3. Design and Development

a. Introduction

This part of the document provides a detailed description of the system. The system shall allow a web administrator to configure the system for obfuscation.

b. Scope

The HTML Obfuscation System is designed to obfuscate the text in an html code so as to render the Information-theft difficult. The application will provide

HTTP Request Handling

TCP Forwarding

HTML Parsing

HTML Obfuscation

Static Content Changing

c. Product Perspective

The information that is displayed in an HTML request via web browser is open source. Its source code is easily available and information e.g. names, phone numbers, Card numbers, blogs, articles etc. can be extracted and used for mal intentions.

The HTML Obfuscation intends to hide such information by making it difficult to locate in the source code. The obfuscated HTML code is equivalent to the original HTML code. The Application works on the following lines

Receive HTTP Request from Browser

Forwards Request to IIS

Receive Response from IIS

Parse, Obfuscate Response

Reply back to Browser

Fig 3.C.1

d. Product Functions

HTTP Request Handling

Brief Description

The Application Handles the HTTP Request made by the browser/client.

Normal Course

- The Application reads the port number from Config File
- The Application Opens the TCP Listener at the specified port
- The Application Listens for incoming requests

- On receiving a request, the Application Opens a new Thread to maintain the communication channel
- Trigger the event to create corresponding IIS connection

Alternate Course

- System shall generates an Error Message for
    - No Config File found
    - Fail to Open the Specified Port
- Request Timeout – Dispose Off the Allocated Resources

IIS Connection Management

Brief Description

For each incoming HTTP Request, corresponding TCP Connection is established with IIS.

Normal Course

- The Application reads the IIS Connection specifications from Config File
- The Application waits for the trigger of new IIS Connection event.
- Once event is fired, the TCP Connection is established with IIS
- The IIS connection is paired with the corresponding thread.
- All the data from client is forwarded to the IIS
- On receipt of Response, Response Received event is triggered

Alternate Course

- System shall generates an Error Message for
    - No Config File found
    - Fail to Established connection with IIS

- Request Timeout – Sends Error message to Client, Dispose Off the Allocated Resources

HTTP Response Handling

Brief Description

The Application receives the HTTP Response from the IIS.

Normal Course

- Once the IIS Connection is established and data is sent to the IIS, The Application waits for the response from IIS
- On trigger of the Response Received Event, the response content is read from the connection
- The content is filtered to separate headers from the payload
- The payload is passed to HTML Parsing module
- On receipt of obfuscated HTML, the HTML headers are added
- The obfuscated response is sent to the Browser/Client.

Alternate Course

- In case of no payload, directly sends the response to client/browser.

HTML Parsing/Obfuscation

Brief Description

Parse the received content for HTML and obfuscate it.

Normal Course

- The module waits for content received event to trigger
- Once content is received, the module parse it to generate HTML tree

- The obfuscation configuration shall be read from config file
- The obfuscated code shall be generated for the specified HTML tags/Text

Alternate Course

- The config file is not found
- Not HTML content – pass it without parsing
- Nothing to obfuscate in HTML

Application Configuration

Brief Description

This file will include the configurations for Parser/Obfuscator, IIS Connection Manager, HTTP response handler, HTTP request handler. It will be updated/set by the admin.

Normal Course

- Admin shall login to the application to access.
- IP:Port for incoming connections
- IP:Port for establishing IIS Connection
- The maximum number of pending connections
- Urls for which obfuscation to be done
- Obfuscation Table (Regular Expressions, Static content replacement etc)

Alternate Course

- Application fails to store/load configuration File, prompt and error message

e. Assumptions and Dependencies

Media Content is not expected to be obfuscated

HTML Obfuscation application shall be installed on web server with full rights to read and write on IO streams.

Performance of the application is directly proportional to web server's processing power and request handling capacity.

f. Quality Attributes

Functionality

The application shall be configurable via its interface. It shall handle the requests, parse the code, match the patterns, obfuscate the information, and generate the response. It shall open a connection with the IIS server to get the html code of the requested web page.

Performance

The application's request handling capabilities are directly proportional to the system it is installed on, how much ram does it have, how many cores of CPU, memory etc. The application shall itself not put constraint on system's resources.

Availability

The system shall be available as long as the system it is installed on is running. This is usually 24 hours as web requests come at any time.

Modifiability

The application shall be easy to modify and update and an improved version shall not involve building the application from scratch.

Portability

The application is designed to run on windows environment but it can handle requests from any kind of system. The IIS server is not the only server that can be connected to it. Any kind of web server can be connected to the application.

Reusability

The application shall be reusable. Each module of the application can be easily incorporated in any other system if need be.

Integrate-ability

All the modules shall be integrate-able to each other and to any similar system as well.

Testability

Different quality test can be performed on the application so as to ensure that application is performing well and without faults.

g. Architectural model

The system will be made using **Event DrivenArchitecture (EDA)** Approach, in which the flow of the program is determined by events. All the HTTP Requests from the Browsers/Client shall be received by the Application at the server. For each incoming HTTP Connection, Corresponding TCP Connection shall be made

with the IIS to forward the received data. On response by the IIS, the Content shall be parsed to obtain HTML. The parsed HTML shall be Obfuscated and sent back to the Browser/Client. The Admin shall configure the Application for Obfuscation and IIS Connection.

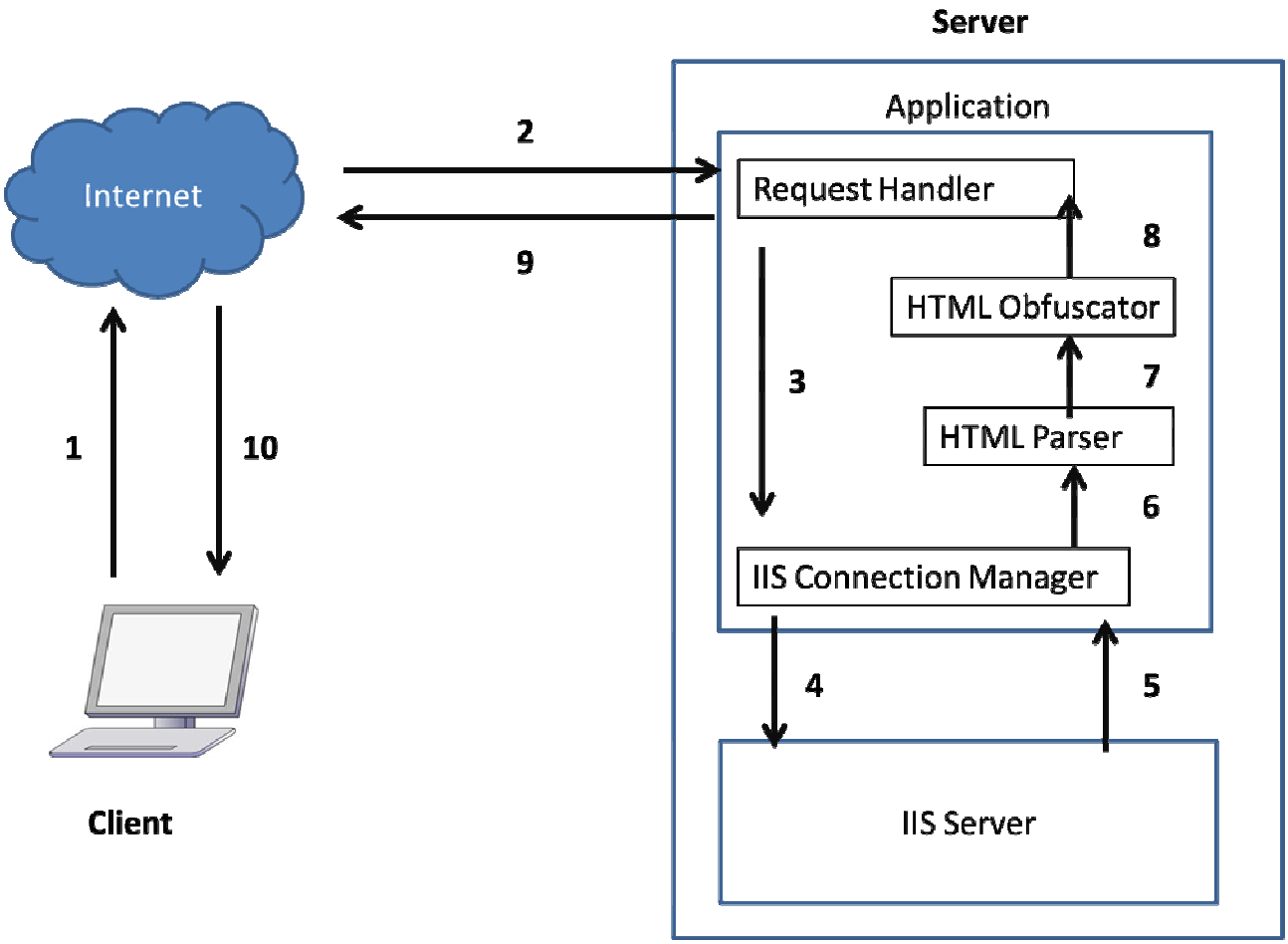


Fig 3.G.1

The reasons to use **EDA** are as follows:

Web based applications are all about user interactions i.e. events.

Each request that is generated at user level, triggers a response from the server.

Each event is independent of each other.

Each event is recognized by its type and parameters and obfuscation accordingly can be provided.

h. Logical View

Logical view contains class diagram and use case diagram. It describes the static behavior of system.

Use case Diagram

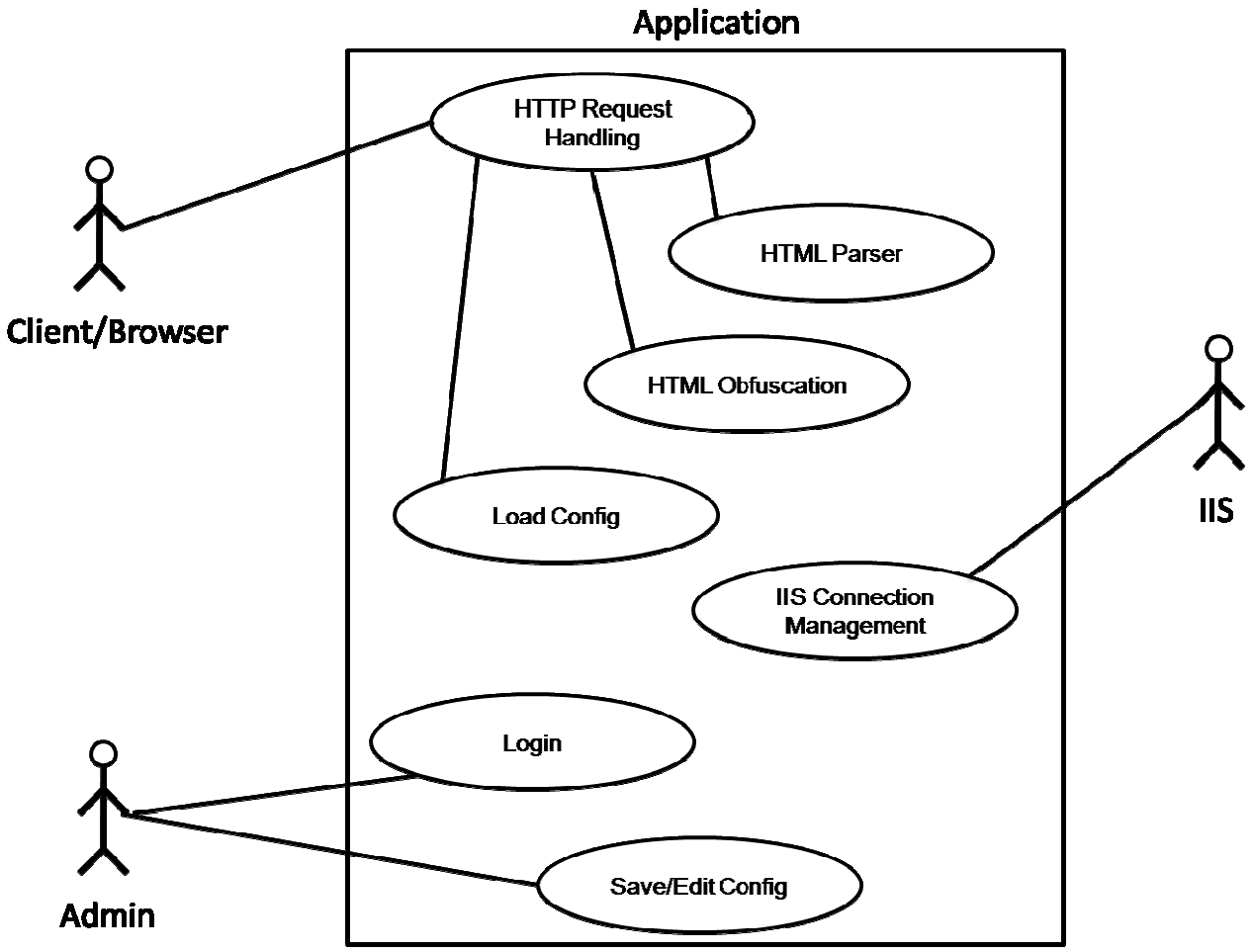


Fig **3.H.1**

Class Diagram

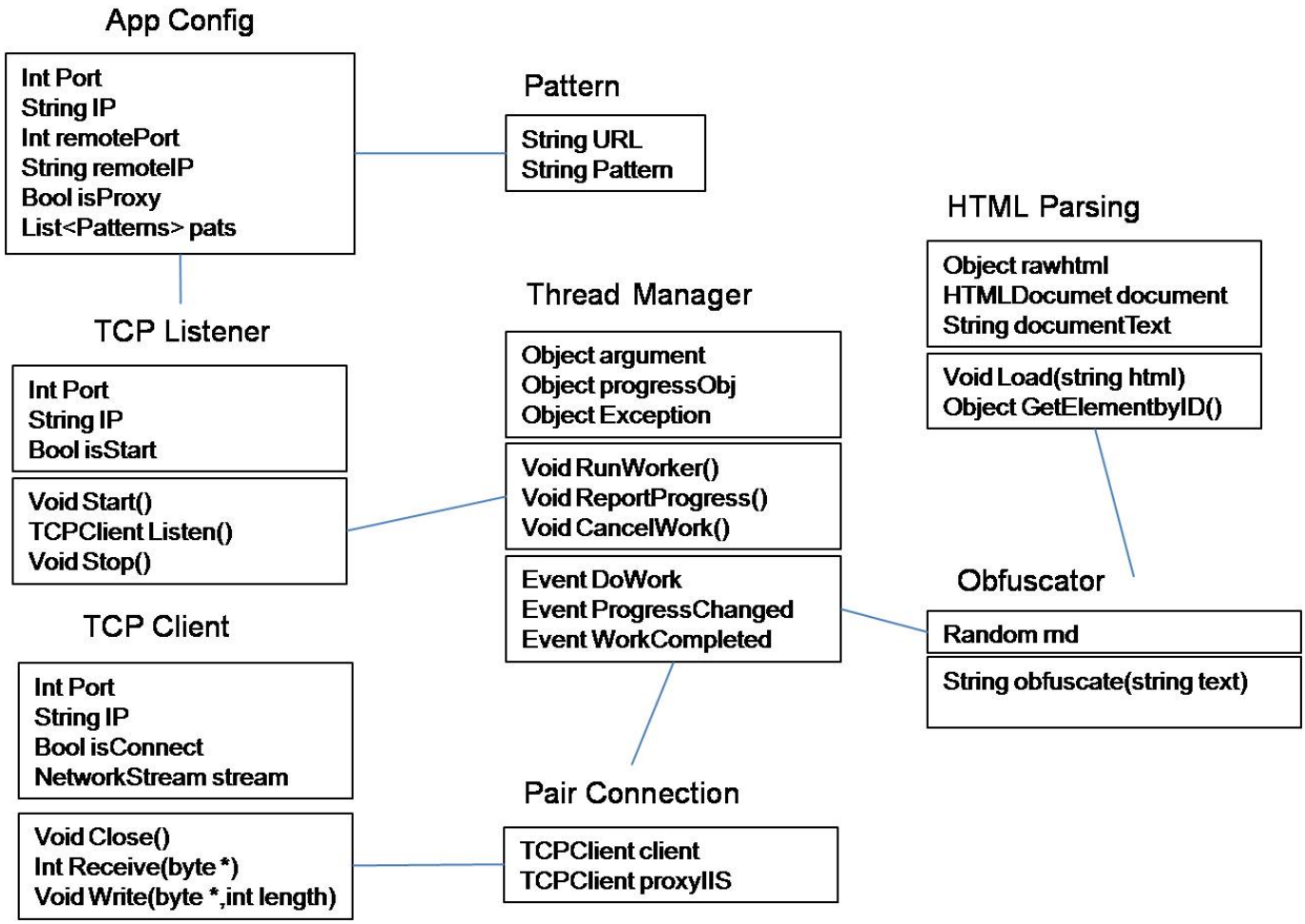


Fig 3.H.2

i. Basic Flow

The application presents with an easy interface where admin can put in patterns of text that needs obfuscation.

Web site owner/author tells the admin to obfuscate certain information over web response.

Now whenever a new web request is received by the server, its original HTML code is sent to parser.

Parser sends the code to obfuscator after parsing.

Obfuscator uses the config file to obfuscate the required kind of text.

Obfuscated code is then sent back to the client.

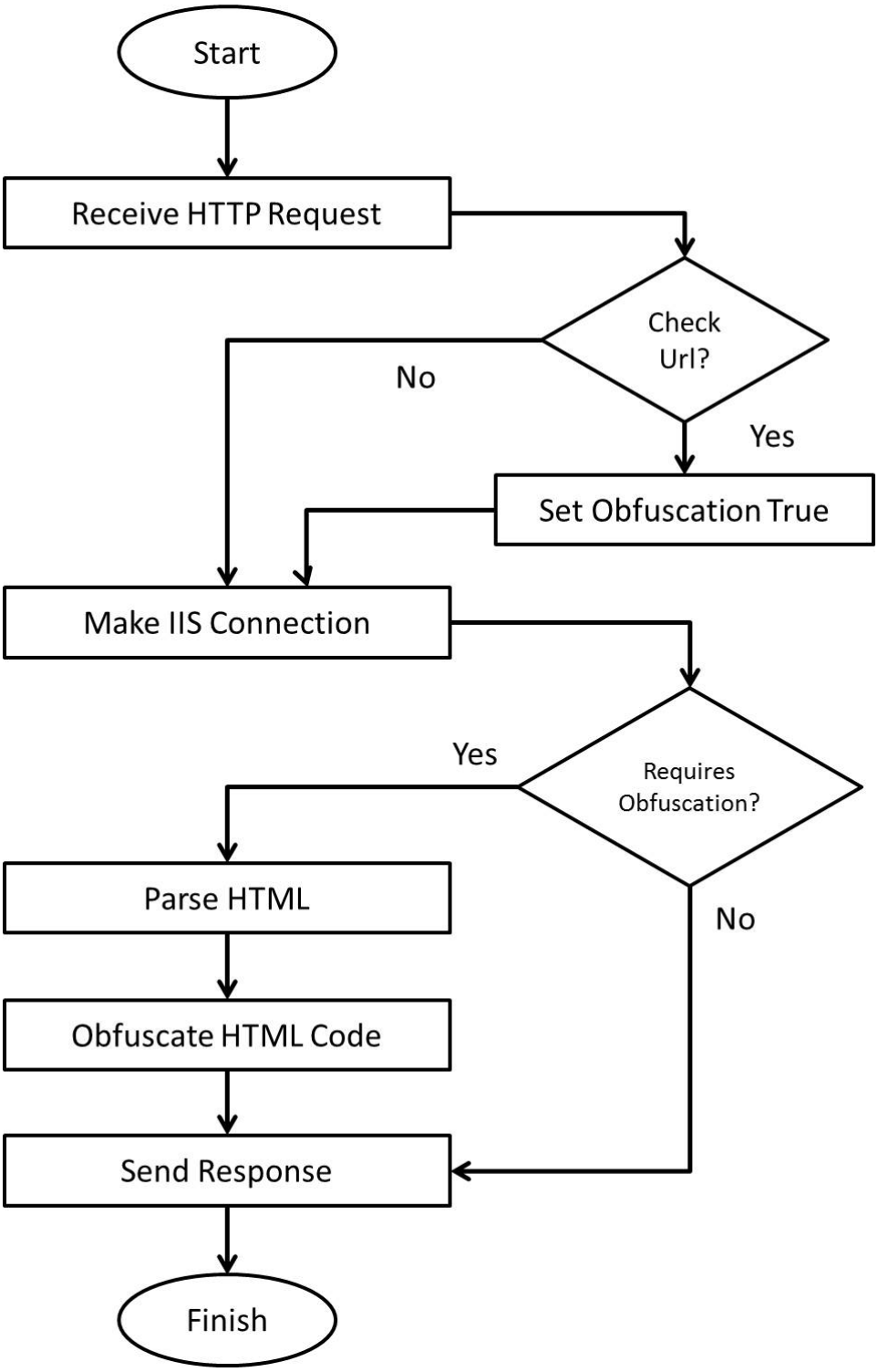

Start
Receive HTTP Request
Check Url?
No
Yes
Set Obfuscation True
Make IIS Connection
Requires Obfuscation?
Yes
No
Parse HTML
Obfuscate HTML Code
Send Response
Finish

Fig 3.I.1

j. Dynamic View

Sequence Diagram

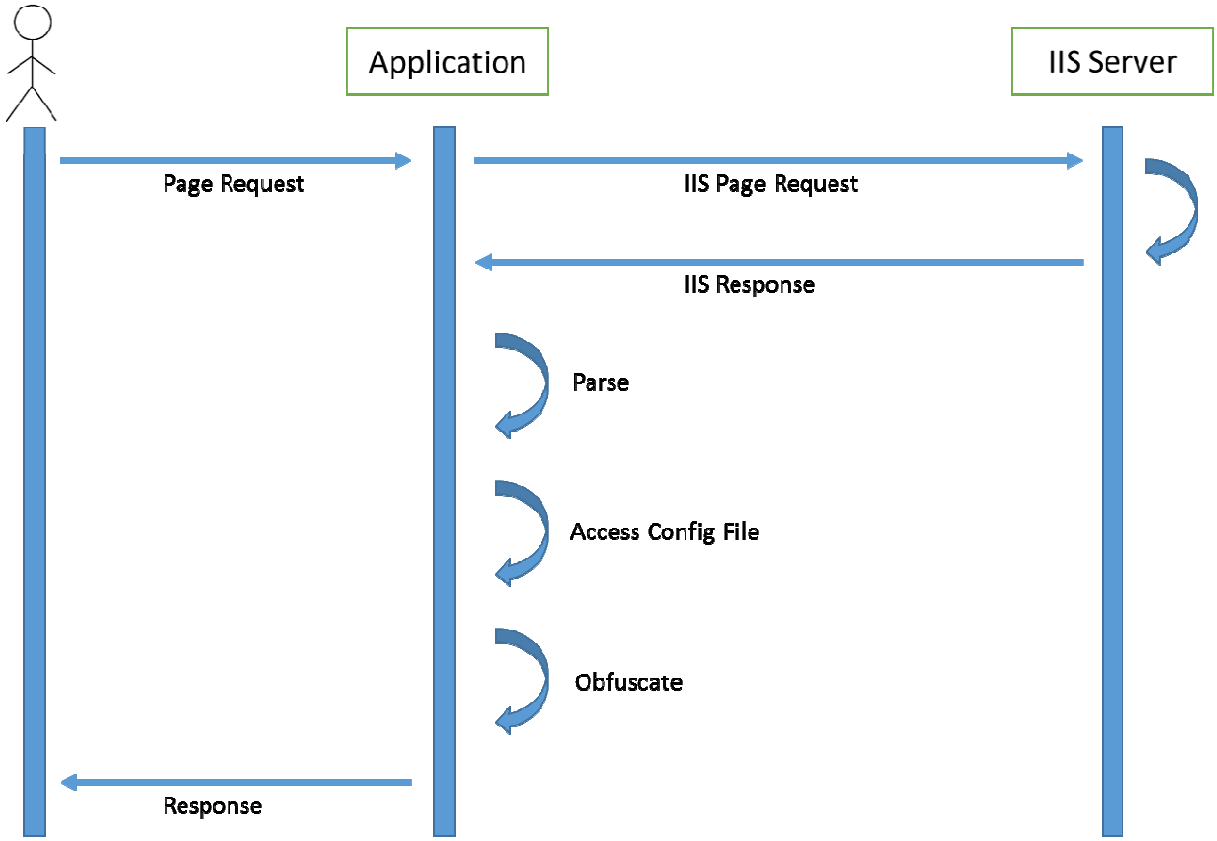


Fig 3.J.1

Data Flow Diagram

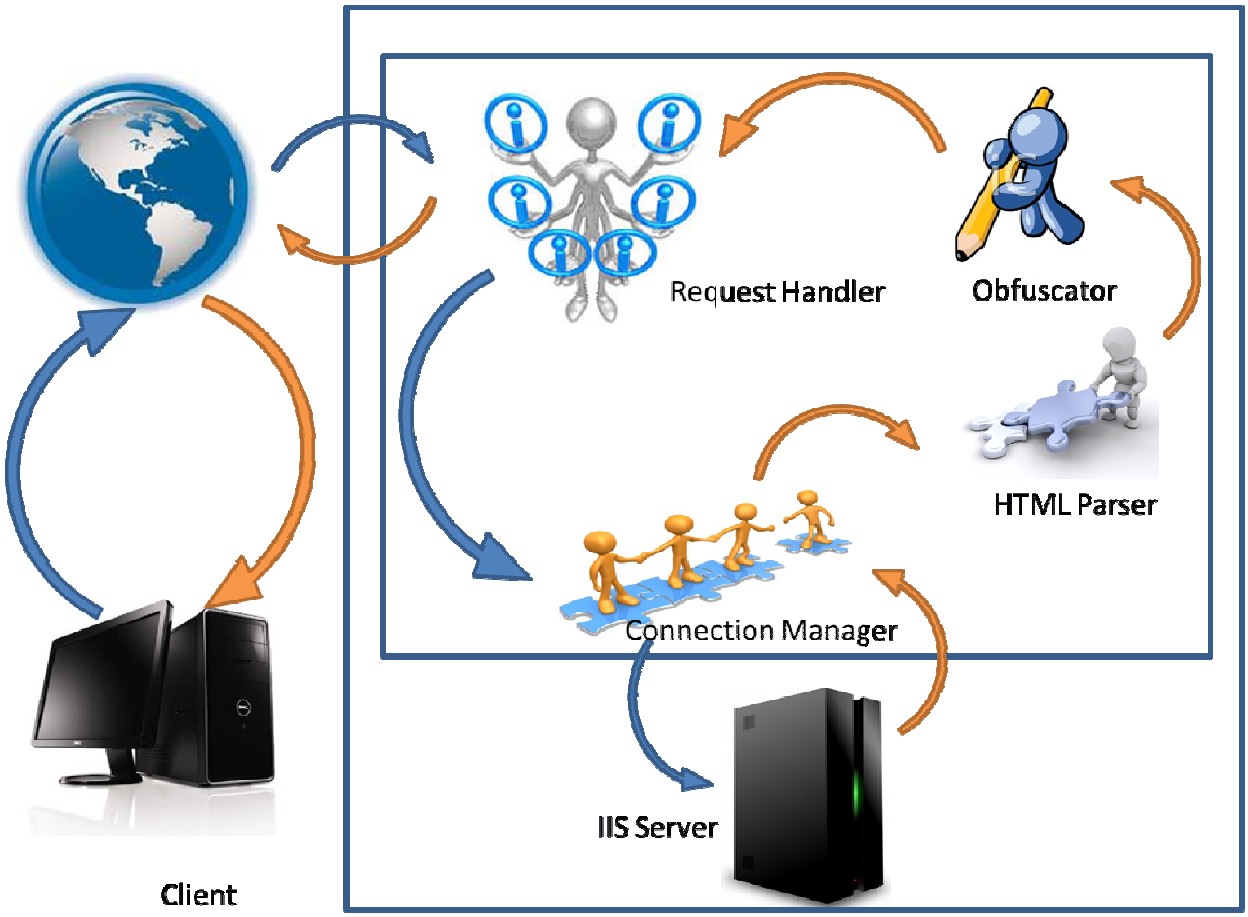


Fig 3.J.2

4. System Implementation Tools and Technologies

a. Microsoft Visual Studio 2008

[1]Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

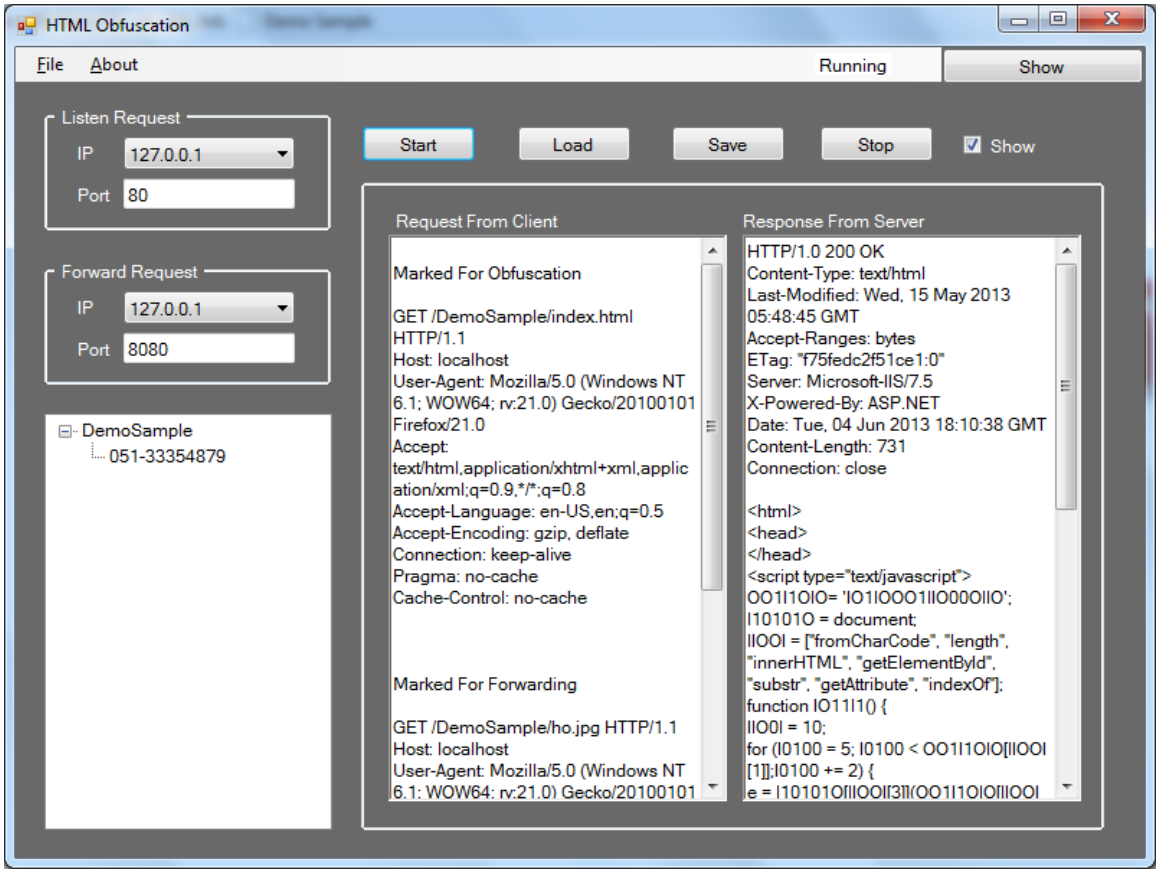
b. C#

[2]C# (Pronounced: C Sharp) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, procedural, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

[1] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio
[2] http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

5. Software Implementation

The software has been using concepts of OOPs with Event driven methodology. All the functionality is implemented in related classes and intercommunication between the classes is performed by passing variables to appropriate functions or notifying to the registered events with related data so that concerned class can deal with the content accordingly. A single class interface “HTML Obfuscator” is provided for the user run the application. A GUI based panel is provided for easy configuration of the software settings.



a. Initialize Settings

The class AppSettings holds all the configuration data of the software. It contains data members for ip and port settings for incoming requests and forwarding requests. A dictionary containing List of urls along with related list of patterns for each url is stored in the AppSettings which is used later on to determine whether to obfuscate a content or not.

```
[Serializable]
        class AppSettings
        {
            public int _rPort = 0;
            public string _rIP = "";
            public int _fPort = 0;
            public string _fIP = "";

            public Dictionary<string, List<string>> _obfUrlPat = new
Dictionary<string, List<string>>();
        }
```

The class is made serializable so that its state can be converted to stream and can be saved to a binary file and later on this state can be read from the file. To achieve this, two methods LoadFromFile and SaveToFile are implemented.

```
static public void SaveToFile(string filename)
        {
           using (Stream stream = File.Open(filename, FileMode.Create))
            {
            BinaryFormatter bFormatter = new BinaryFormatter();

            bFormatter.Serialize(stream, _appSetings);
            }
        }
```

The SaveToFile method takes one string argument for filename, to which the state of the variable is to be stored. The object of AppSettings class is serialized with binary formatter, and the output stream is written to the file.

```
static public void LoadFromFile(string filename)
        {
           using (Stream stream = File.Open(filename, FileMode.Open))
            {
            BinaryFormatter bFormatter = new BinaryFormatter();

            _appSetings = (AppSettings)bFormatter.Deserialize(stream);
            }
        }
```

The LoadFromFile method, opens the stream to the file, specified in the method argument. The stream is de-serialized with the binary formatting and stream is type casted to the AppSettings class, thus restoring previously stored state to a variable of the AppSettings class.

Properties are provided to access the data members of the AppSettings class for directly configuring their values.

b. Running the Application

This is the main class which communicates with all the remaining classes and provides a single point of interaction to the user. User just needs to load the configurations either from file or by directly setting the properties of the AppSettings class. Two most important data members of the class are _settings variable of the type AppSettings and _server variable of the type server. _settings variable holds all the configuration of the software, whereas _server uses some of the configurations to listen for incoming requests from the client.

```
static class HTMLObfuscator
    {
      static AppSettings _appSetings = new AppSettings();

      static Server _server = new Server(); . . .
```

Once the configurations has been done i.e. IP,port for receiving and forwarding along with the url and pattern dictionary, user can run the application by calling the Start method of the HTMLObfuscator class, which calls the start method of the underlying server class.

```
static public void Start()
```

The class has a data member of data type Server, once the start method is called, the server starts listening for the incoming request at the specified IP,port. To stop the server at any point, user can call the Stop method of the HTMLObfuscator class.

```
static public void Stop()
```

Various methods are implemented to easily control the behavior of the software at run time. Following of the methods changes the configurations of the software.

```
static public void SetIpPorts(string ReceiveIP, int ReceivePort, string ForwardIP, int ForwardPort)
```

The above method changes the current ip port settings to the passed argument values. Following Add, Remove methods provide functionality for changing the urls and their patterns at run time.

```
static public void AddURL(string url)

static public void RemoveURL(string url)

static public void AddPattern(string url,string pattern)
```

```
static public void RemovePattern(string url, string pattern)

static public void AddURLPatterns(string url, List<string> patterns)

static public List<string> GetPatternsForUrl(string url)
```

To update the user interface with the requests of client and response to the client, an event is implemented which passes data to the GUI class. The two variables passed in the event define the type of the event and related data with the event.

```
public delegate void Update(string code, string data);
static public event Update UpdateToInterface;
```

To receive updates from the HTMLObfuscator class, the GUI class needs to register to the Update event and delegate a method which will be called upon once the event is fired. This is done in GUI class as following

```
HTMLObfuscator.UpdateToInterface += new
HTMLObfuscator.Update(HTMLObfuscator_UpdateToInterface);

void HTMLObfuscator_UpdateToInterface(string code, string data)
```

c. Incoming Connections

Server class implements the logic for listening to incoming requests. A TcpListener class object is made on instantiation of the object of Server class. On calling start method of the server class, the TcpListener object is bound to incoming request ip, port from AppSettings.

```
_server = new TcpListener(IPAddress.Parse(HTMLObfuscator.RecieveIP),
HTMLObfuscator.RecievePort);
```

A background worker thread is created to listen for incoming requests so that TcpListener object doesn't block the main thread to respond to the user input.

```
while (true)
{
	try
	{
		TcpClient client = _server.AcceptTcpClient();
		bgw.ReportProgress(0, client);
	}
	catch (Exception ex)
	{
		break;
	}
}
```

The server receives the requests from the clients, and then passing the requests to the ThreadManager class for further processing. When so ever a new request is received from client, the class makes a new forwarding connection with the web server i.e. IIS in our case, make a connection pair with the incoming connection and forwarding connection so that data between these two can be easily exchanged.

```
TcpClient fwdConn = new TcpClient(HTMLObfuscator.ForwardIP,
HTMLObfuscator.ForwardPort);

ConnectionPair conP = new ConnectionPair(client, fwdConn);
```

The ConnectionPair class holds two data members of TcpClient type, one for client and one for the web server. Then this connection pair is passed to the ThreadManager class which does the further processing on the client request.

```
ThreadManager thM = new ThreadManager(conP);
thM.Manage();
```

d. Dealing with Requests

The ThreadManager class is responsible for identifying the type of request and then deciding whether to pass this request to ObfuscationThread or ForwardingThread. This is done by obtaining the HTTP request header from the client connection and then parsing it. The ThreadManager class opens the NetworkStreams to the client and forwarding connection based upon the ConnectionPair object passed it to by the Server class.

```
ConnectionPair conP;

NetworkStream nsC = conP.nsC;

NetworkStream nsP = conP.nsP;
```

Once streams are open, the data is read from the client stream into a byte array.

```
byte[] data = new byte[8192];
int recv = 0;
try
{
recv = nsC.Read(data, 0, data.Length);
}
```

Received bytes are encoded to get ASCII Encoding to get the string representation of the received bytes.

```
string _httpReqHeader = Encoding.ASCII.GetString(data, 0, recv);
```

HTTP Request Header is parsed to obtain the url of the request. This url is checked in the dictionary of AppSettings to decide whether this request to be marked for forwarding or not. If url exists in the dictionary's url list, then this

request is passed to ObfuscationThread otherwise ForwardingThread is made responsible for rest of the task.

```
List<string> pats = new List<string>();
      foreach (string u in HTMLObfuscator.AllUrls)
      {
      if (url.Contains(u.ToLower()))
      {
            isObf = true;
            pats = HTMLObfuscator.GetPatternsForUrl(u);
            break;
            }
      }
```

Once decision for the thread has been made, ThreadManager class writes the HTTP Request Header data to the forwarding port so that web server IIS, can generate the appropriate response.

```
      data = Encoding.ASCII.GetBytes(msg);

      nsP.Write(data, 0, data.Length);
```

The response from the IIS, on the way back will be handled by the ObfuscationThread or ForwardingThread class based upon the condition to which thread the HTTP Request was assigned.

e. Dealing With Response

Once the web server sends the response of the request to the application, one of the class ObfuscationThread or ForwardingThread, will deal with it. In case ForwardingThread, the response is just read from the forwarded stream and write it to the client stream with no modification. Each time a new connection pair is passed to the ForwardingThread class, a separate thread is created to perform the task. The response data is read from the web server stream into byte arrays,

and then these bytes are written to the client stream. On completion of the data, the streams are flushed and closed.

```
while (true)
      {
            byte[] b = new byte[10000];
            int recv = conP.nsP.Read(b, 0, b.Length);
            if (recv > 0)
            {
                  conP.nsC.Write(b, 0, recv);
            }
            else
                  break;
      }
```

In case of ObfuscationThread, the response is to be parsed, obfuscated and then sends back to the client by writing it on client stream. Once the ObfuscationThread is started, it waits for the receipt of data on the forwarding stream i.e. stream of the connection with web server. Once data receipt event is fired, the HTMLParser class is assigned the responsibility to read the data from the stream and return an object of HTMLPacket type.

```
HTMLPacket hp = HTMLParser.ParseFromStream(conP.nsP);
```

The HTMLParser class reads the bytes from stream and determines the headers fields and look for the end of the HTTP header so that remaining bytes can be taken as HTTP content.

```
      StreamReader nsP = new StreamReader(ns, Encoding.ASCII);

      string header = "";

      while (true)
            {
                  string txt = nsP.ReadLine();
                  if (txt != "")
                  {
                        header += txt + "\r\n";
                  }
```

In this regard content length field is determined from HTTP Header fields to decide the end of data on stream and keep integrity of the content of the HTTP response. An object of HTMLPacket type is made and header information is passed into it. This class split the header into HTTP Header fields and determines the length of the content. Content length property is used to determine how much bytes of data on stream are left.

```
Regex reg = new Regex("Content-Length: [0-9]+");
Match m = reg.Match(_header);
if (m != null && m.Value != "")
{
_length = long.Parse(m.Value.Replace("Content-Length: ", ""));
}
```

Once content length is calculated, the HTMLParser class reads the remaining bytes from the stream, encode it to characters and put it as content in the HTMLPacket type object.

```
char[] data = new char[_htmlPacket.ContentLength];

int completed = 0;

while (completed < _htmlPacket.ContentLength)
      {
      char[] buffer = new char[8192];
      int recv = nsP.Read(buffer, 0, buffer.Length);
      Array.Copy(buffer, 0, data, completed, recv);
      completed += recv;
      }
_htmlPacket.Content = new string(data);
```

This HTMLPacket object i.e. HTTP Response, is returned to the ObfuscationThread class. Then based upon the url in the header fields of the HTTP response, the ObfuscationThread class pulls the list of patterns from dictionary of the AppSettings class and passes the list of patterns along with the content of the HTTP Response to the ContentObfuscator class which returns the obfuscated content for the given patterns.

```
ContentObfuscator cObf = new ContentObfuscator();
hp.Content = cObf.Obfuscate(hp.Content, patterns);
```

The obfuscated content along with corresponding HTTP response Header is converted into bytes and sent to the client.

f. Obfuscating the Response

Content Obfuscator class deals with the obfuscation of the content for the given patterns. Obfuscation is done by adding a JavaScript function in the content which is dynamically created at run time. The patterns which are to obfuscate are iterated and their matches are found in the content. For each match found, the match is replaced by a span tag. This span tag is assigned an id and tag value. The id of the span tag is later on used by the JavaScript function to find these span tags so that original content can be put in these span tags. The tag field of the span tags helps in determining what content to be replaced. The JavaScript function holds few variables to determine the ids and tags of the span fields so that once this JavaScript function is executed in client browser it can find the span tags. The names and values of these variables are changed on each request. Moreover the pattern of the matches found is broken by randomly

splitting and padding extra text. The whole JavaScript function uses difficult to understand variable and function names built up with characters I, O, l, 1, 0.

```
char[] obfChars = new char[5] { 'I', 'l', '1', 'O', '0' };
private string GetRandomLengthObfString(int length)
  {
      StringBuilder str = new StringBuilder();
      for (int i = 0; i < length; i++)
      {
          str.Append(obfChars[rnd.Next(0, 5)]);
      }
      if (str[0] == '0' || str[0] == '1')
      {
          str[0] = 'I';
      }
      return str.ToString();
  }
```

The GetRandomLengthObfString method takes an integer argument and returns a string made of obfuscated characters of the passed length. String returns from calls to this function are used as variable names, ids and tags for the span tags. A list of name of JavaScript functions, which are used in obfuscated function, is maintained and shuffled each time a request for obfuscation comes.

```
List<string> obfFnNames = new List<string>() { "getElementById",
"getAttribute", "substr", "indexOf", "fromCharCode", "length",
"innerHTML" };
```

```
public void Shuffle(IList<string> list)
  {
      var randomNumber = new Random(DateTime.Now.Millisecond);
      var n = list.Count;
      while (n > 1)
      {
          n--;
          var k = randomNumber.Next(n + 1);
          var value = list[k];
```

```
                list[k] = list[n];
                list[n] = value;
            }
        }
```

This shuffle helps in generating a different obfuscated pattern for the same pattern on each request. To avoid replacing similar matches of a pattern with same obfuscated value, a replace first method is implemented which only replaces the first occurrence of the pattern from the given index.

```
public string ReplaceFirst(string text, string search, string replace)
      {
      int pos = text.IndexOf(search);
      if (pos < 0)
      {
            return text;
      }
      return text.Substring(0, pos) + replace + text.Substring(pos +
search.Length);
      }
```

The obfuscate method, generates all the necessary code of JavaScript language. It replaces the matches with span tags with random ids and random tag values. It makes use of above method functions to perform the task. Following piece of code perform the search of patterns in the content, generating the random ids and their tag values and replacing the matches for patterns with span tags

```
foreach (string pat in pats)
{
      Regex reg = new Regex(pat);
      MatchCollection mc = reg.Matches(content);
      foreach (Match m in mc)
      {
      rndID = baseStr.Substring(baseStr.Length - maxIDsLim);
      rndTag = baseStr.Substring(baseStr.Length - maxTagsLim);
```

```
        baseStr += GetRandomLengthObfString(maxStep);
        content = ReplaceFirst(content, m.Value, "<span id='" + rndID +
"' " + rndTag + "='" + EncodeTo64(rndTag.Substring(0, maxTagMix) +
m.Value + rndID.Substring(0, maxIDsMix)) + "' /></span>");
        }
}
```

Once all the matches have been replaced by tags, a JavaScript is produced which will iterate through all these tags and put back the original match value in its place. In the end, the JavaScript function is made to run on onload event of the body tag of the content.

```
script += "</script>" + Environment.NewLine + "<body onload=\"" +
onLoadFnName + "\"";
return content.Replace("<body", script);
```

```
 8 function IO1111() {
 9 IlO0l = 10;
10 for (I0100 = 5; I0100 < OO1I1OlO[lIOOI[1]];I0100 += 2) {
11 e = l101010[lIOOI[3]](OO1I1OlO[lIOOI[4]](I0100, 10));
12 x = atob(e[lIOOI[5]](OO1I1OlO[lIOOI[4]](IlO0l, 5)));
13 e[lIOOI[2]] = x[lIOOI[4]](2, x[lIOOI[1]] - 6);
14 IlO0l+= 2;
15 }
16 }
17 </script>
18 <body onload="IO1111()">
```

6. Project Analysis and Evaluation

a. Testing

To ensure quality of the product, testing is conducted. Accuracy and efficiency of tasks performed by our system had to be tested to analyze the system and verify and validate it. Software testing techniques and results obtained are discussed in the coming sections.

b. Testing Levels

Separate modules were developed to provide different functionalities of the system. All of these modules were tested at different levels during development and after integration. Different levels of testing and results have been described here:

I. Unit Testing

Each module was designed, developed and tested individually. Each functionality was also tested separately.

i. HTTP Request Handling was tested to ensure that all requests were going through this module. This module listens on port 80 (or whichever port is set for web requests).

ii. IIS Connection manager was tested to ensure it establishes a request-response connection with the server. The server is set to listen on the same port as this module so as to allow the flow of data.

iii. HTTP Response Handler was tested to see if it is sending the processed response over on the internet to the user.

iv. HTML Parser was tested to see if it generates a valid parse tree which can then be used to traverse through the information/code easily.

v. Obfuscator was tested to see if it recognizes which all text was required to be obfuscated. It then was tested to see if it was able to generate random java script function and variables every time.

Test Case ID	1
Unit to Test	Request Handling
Assumptions	1. LAN established 2. IIS application is running 3. "HTML Obfuscation" is running
Test Data	1. HTTP Request Headers

Steps to be Executed	1. Client sends request for web page to server (port 80). 2. Application on server listens the request on port 80. 3. Request forwarded to IIS over port 8080. 4. Use traffic sniffer (WireShark) to monitor traffic and confirm flow of HTTP request.
Expected Result	Request flows from client to application and then to IIS.
Actual Result	As Expected
Pass/Fail	Pass

Test Case ID	2
Unit to Test	HTML Parser
Assumptions	1. LAN established 2. IIS application is running 3. "HTML Obfuscation" is running
Test Data	1. HTTP Parse trees

Steps to be Executed	1. Client sends request for web page to server (port 80). 2. Application on server listens the request on port 80. 3. Request forwarded to IIS over port 8080. 4. Response from IIS is received by HTML Parser. 5. For test purposes, Test Tree was printed on the screen for every request
Expected Result	Valid Parse tree is made.
Actual Result	As Expected
Pass/Fail	Pass

Test Case ID	3
Unit to Test	Obfuscator
Assumptions	1. LAN established 2. IIS application is running 3. "HTML Obfuscation" is running
Test Data	1. HTTP Source Code's Parse Tree

Steps to be Executed	1. Client sends request for web page to server (port 80). 2. Application on server listens the request on port 80. 3. Request forwarded to IIS over port 8080. 4. HTML Parser generates the parse tree. 5. Obfuscator Traverses through the parse tree and matches the patterns. It then obfuscates and generate equivalent code. 6. Obfuscation is confirmed at the client end observing the source code of the web page.
Expected Result	Obfuscated code is generated at random on every request
Actual Result	As Expected
Pass/Fail	Pass

II. Integration Testing

vi. Initially Request handler and Response handler were tested for integration

vii. Then IIS Connection manager was integrated and tested to see if all the requests were flowing properly through application->Server->application.

viii. HTML Parser was integrated next and was tested if it was generating the valid tree to the received html code from server.

ix. In the end, Obfuscator was made and integrated in the application.

III. System Testing

x. System testing was performed at the end of development. Complete system was tested by hosting different kind of web sites and then testing the whole system for performance and other attributes (failures, response delays, connection losses etc).

c. Results

The results of the tests were in the acceptable range. There were very less connection losses , very less application failures (almost none). Although the performance does depend on the system used, there is still margin of

improvement. Here are the results that were achieved after requests were processed through multiple threads.

Sequential Requests – One Thread

Number of Requests	**IIS Total Response Time (ms)**	**App Total Response Time(ms)**	**Difference (ms)**	**Average Processing Delay per Request (ms)**
1000	1563	2046	483	0.483
5000	7560	10074	2520	0.504
10000	15698	20648	4950	0.495

Parallel Requests – 10 Threads

Number of Requests	**IIS Total Response Time (ms)**	**App Total Response Time(ms)**	**Difference (ms)**	**Average Processing Delay per Request (ms)**
1000	1015	1394	379	0.379
5000	5375	6860	1485	0.297
10000	11222	14442	3220	0.322

Parallel Requests – 50 Threads

Number of Requests	IIS Total Response Time (ms)	App Total Response Time(ms)	Difference (ms)	Average Processing Delay per Request (ms)
1000	1051	1506	455	0.455
5000	5575	6860	1285	0.257
10000	11281	14495	3214	0.321

d. Analysis

The results tell us that the application is truly implementing multi threading. This means that the dependency of performance is directly proportional to the kind of processor, number of cores, available memory. Usability of this application makes it easy to be re-used and implemented /integrated in other. The learning curve is short.

This application implements a new idea that is emerging. Dynamic obfuscation can be further implemented in many future projects.

7. Conclusion and Future Work

The goal of this project was to implement HTML obfuscation dynamically over a web-server. The project was chosen after careful selection mainly because very less work has been done on this and also because it was interesting. The team had set an aim to change the static implementation of HTML obfuscation into a dynamic one. The goal was set and achieved as per planned timeline.

During the course of development, the team encountered some difficulties, especially when it came to "Obfuscator" module. The team had some experienced in C# but it needed polishing. There were no binaries/Dynamic Link Lists or reusable classes available. This meant developing / planning / tweaking everything from scratch. Also JavaScript was needed to be learnt. The team also required in-depth knowledge of HTML requests and responses, their headers and their flow over the internet.

The goal (thanks to Allah Almighty) has been achieved till now, although the team shall continue its work on this application in their own capacity to make it more efficient and also implement this idea into other applications. The team hopes that this project brings good name to Military College Of Signals, NUST and the Armed Forces of Pakistan. "Amin!"

Appendix A: Glossary

HTML – Hyper Text Markup Language

IIS – Internet Information Server

Config – Configuration File

Appendix B: References

http://en.wikipedia.org/wiki/Obfuscation_%28software%29

http://www.albertawebsitemarketing.com/what-is-html-obfuscation-and-why-should-web-designers-care-about-it/

http://colddata.com/developers/online_tools/obfuscator.shtml

http://htmlobfuscator.com/

http://colddata.com/developers/online_tools/obfuscator.shtml

http://www.wmtips.com/tools/html-obfuscator/

This was a desktop application that was already developed

http://www.softpedia.com/get/Internet/WEB-Design/Source-Site-Protectors/HTML-Obfuscator.shtml