WEB CONTENT SCREENER

PROJECT REPORT

P-1176

Submitted in partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY

OF THE BHARATHIAR UNIVERSITY, COIMBATORE

Submitted by

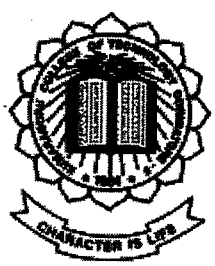
B.ANANTH (0027S0062)

A.ARUN PRAKASH (0027S0067)

A.SATISH KUMAR (0027S00103)

Under the valuable Guidance of

Ms.N. Rajathi , B.E.,
SENIOR LECTURER, IT DEPARTMENT



MARCH 2004

Department of Information Technology

Kumaraguru College of Technology

(Affiliated to Bharathiar University)

COIMBATORE – 641 006

KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to Bharathiar University)
COIMBATORE – 641 006, TAMILNADU, INDIA
Approved by AICTE, New Delhi - Accredited by NBA

Department of Information Technology

CERTIFICATE

This is to certify that the project entitled

"WEB CONTENT SCREENER"

has been submitted by

B.Ananth, A.Arun Prakash and A.Satish Kumar

in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering Information Technology of the Bharathiar University, Coimbatore – 641 046 during the academic year 2003-2004

Head of the Department

Project Guide

Submitted for the university examination held on 26/3/04

Internal Examiner

26/3/04

External Examiner

Declaration

We,

B.ANANTH	**0027S0062**
A.ARUN PRAKASH	**0027S0092**
SATHISH.A.KUMAR	**0027S00103**

declare that the project entitled "WEB CONTENT SCREENER" is done by us and to the best of our knowledge. A similar work has not been submitted earlier to the Bharathiar University or any other institution, for fulfillment of the requirement of the course study.

This project report is submitted on the partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology from Bharathiar University.

Place: Coimbatore.

Date :

B. Ananth

[B.ANANTH]

[A.ARUNPRAKASH]

[SATHISH.A.KUMAR]

Project Guided by

Senior Lecturer Miss. N. Rajathi

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We are greatly indebted to our revered Principal **Dr.K.K.Padmanabhan, Ph.D.**, who has been the motivating force behind all our deeds.

We earnestly express our sincere thanks to our beloved Head of the Department Prof. **Dr.S.Thangasamy, Ph.D.**, for his immense encouragement and help and for being our source of inspiration all through our course of study.

We are much obliged to express our sincere thanks and gratitude to our beloved guide Senior Lecturer **Ms.N.Rajathi., B.E.**, for her valuable suggestions, construction criticisms and encouragement which has enabled us to complete our project successfully.

We gratefully thank our Project Coordinator **Mrs.S.Devaki., B.E.,M.S.** and our Class Advisor **Miss P.Sudha, B.E.**, for extending their most appreciative and timely help to us.

We also thank all the staff members of the Department of Information Technology for all their encouragement and moral support.

We also extend our heartiest thanks to all our friends for their continuous help and encouragement throughout the course of study.

SYNOPSIS

SYNOPSIS

Our Project entitled '**WEB CONTENT SCREENER**' aims at controlling Internet access policies tailored to the specific needs of the user. This software provides comprehensive web page filtering based on keywords,url and filetypes such as cookies. This software also removes certain content displayed in a page which includes images, popup windows, ads and banners which are quite annoying to the web users.

The content screener software can be customized to add or remove urls, keywords and strings to be searched for blocking ads and banners. For each feature we maintain an individual access list which can be updated whenever required. Thus this software acts as an application gateway thereby allowing/restricting web access to the users.

CONTENTS

CONTENTS

INTRODUCTION

INTRODUCTION

Web is a repository of information both technical and general which can be shared between users. Internet has witnessed vast scientific advancements right from its inception. People have started using it to a very great extent. It has now become an indispensable part of today's modern life. The web pages that we see these days are dynamic, hence changing their content in minutes. They contain attractive elements like images, multimedia content etc. But there are certain websites which contain unethical content which should not be viewed by certain category of people and those which may take up the resources of the computer resulting in slow access to web pages. Thus we have recognized the need to develop a software which allows/blocks web content pertaining to the needs of the user.

The **'WEB CONTENT SCREENER'** that we have developed will act more or less like a firewall since it allows or blocks traffic based on some rules defined by the user. There are several classifications of firewalls depending on which layer of the OSI model the firewall works. This software comes under the classification of an application gateway since it analyses the data at the HTTP (a protocol working at the application layer) level bringing context information into the decision process.

An application gateway usually works by breaking the client/server model. Every client/server communication requires two connections: one from the client to the application gateway and the other from the application gateway to the server. Thus an application gateway has more control over the context of communication.

There are several commercial firewalls available in the market for both Linux and Windows. Linux by itself has a firewall called iptable which is a packet filter. But there is no software which does web content screening explicitly for Linux as such. That's why we ended up in developing a HTTP gateway which is very easy to use and has a good scope for future enhancements.

SOFTWARE REQUIREMENT SPECIFICATIONS

Software Requirement Specifications

2.1 Purpose:

Our project **"WEB CONTENT SCREENER"** aims at controlling web access policies tailored to the needs of the user. This application provides comprehensive web page filtering based on content and file types.

2.2 General Description:

2.2.1 Product Perspective :

Linux does not have a web content screening software as such.This resulted in the need to develop this application which acts at the application layer and blocks traffic depending on the contents of the web pages.

2.2.2 User Characteristics:

The user should have a basic knowledge of the working environment in Linux. The user must also know how to configure the browser to connect to a proxy.

2.3 Specific Requirements:

2.3.1 Functional Requirements:

2.3.1.1 Introduction

The basic function of this application is to block web pages or its contents depending on the configuration set by the user.The user has provisions to enable/disable a feature.There is a need for a HTTP proxy to enable this application to work.

2.3.1.2 List of Inputs:

- Feature to be blocked/allowed
- Access list strings

2.3.2 Design Constraints:

2.3.2.1 Software Requirements:

The application we have designed requires the following software

1) Red Hat Linux 7.0 or higher
2) Squid Proxy
3) Gnu C compiler
4) A Browser with proxy support.

2.3.2.2 User Interface:

Since the application is to be developed in c, the user interface will only be a command prompt. All the options that are given will be enabled using only commands.

WEB CONTENT SCREENER ARCHITECTURE

WEB CONTENT SCREENER ARCHITECTURE

The Architecture of the **'WEB CONTENT SCREENER'** application is similar to that of an application gateway. The screener acts as a bridge between the client browser and the proxy server intercepting all the requests sent by the browser thereby having complete control over the requests sent by the browser.

First the browser is configured to redirect all the requests to the screener. The screener waits for a request from the client browser. When a request arrives it is first processed. There are two possibilities that can occur after processing the request.

i) If there are undeniable contents in the request, we make a connection with the proxy server requesting it to fetch the page from the web server.The fetched page is then scanned for any contents that has been requested to be blocked by the user. If found those contents are blocked or else the requested page is sent back to the client browser.

ii) If the browser request has undesirable contents, then we send a negative reply for the corresponding request from the client browser that the requested content cannot be fetched.

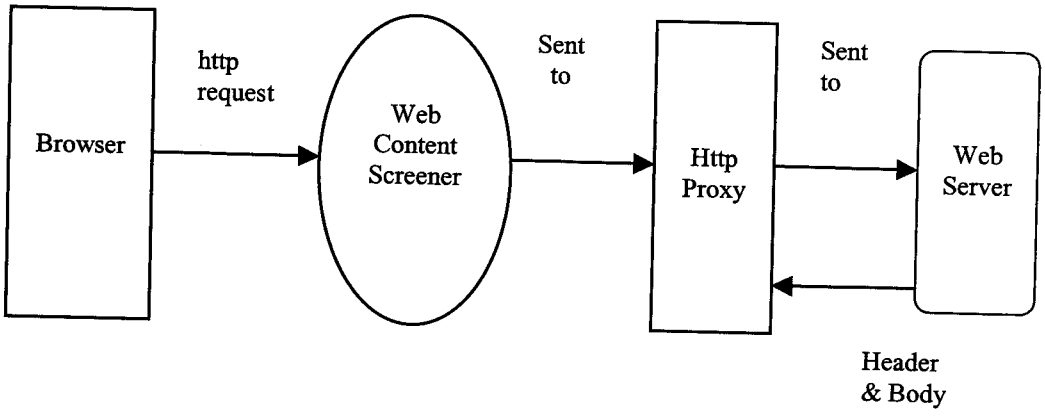
By this process we provide a complete control over the context of the communication. Only limited amount of contents are getting downloaded, so the bandwidth and time gets saved.
The whole process is done in three steps:

1. Request Parsing
2. Content Parsing
3. Returning screened Page

At the end of these three steps the client browser can get a denied page or the requested page. Now let us see these three steps in detail

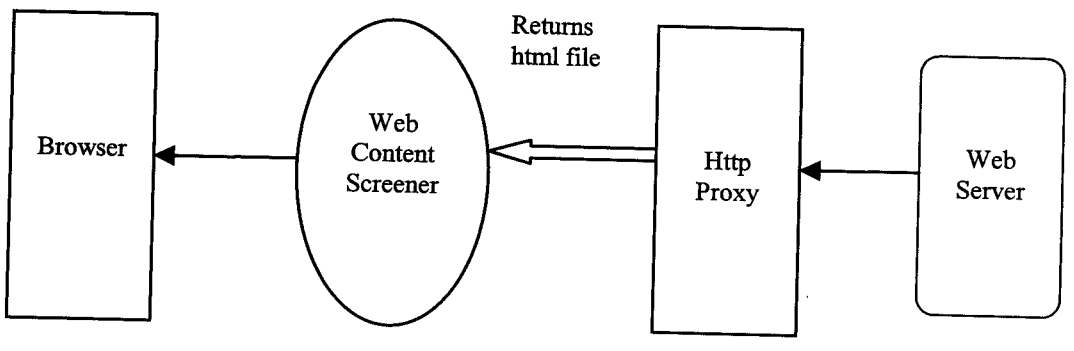
1. Request Parsing:

The client browser makes a request to the screener which checks therequest for any URL match from the list provided by the user. If there is a URL has been found in the list then the custom page is sent. If there is no match then the browser request is then sent to the proxy server for further processing.



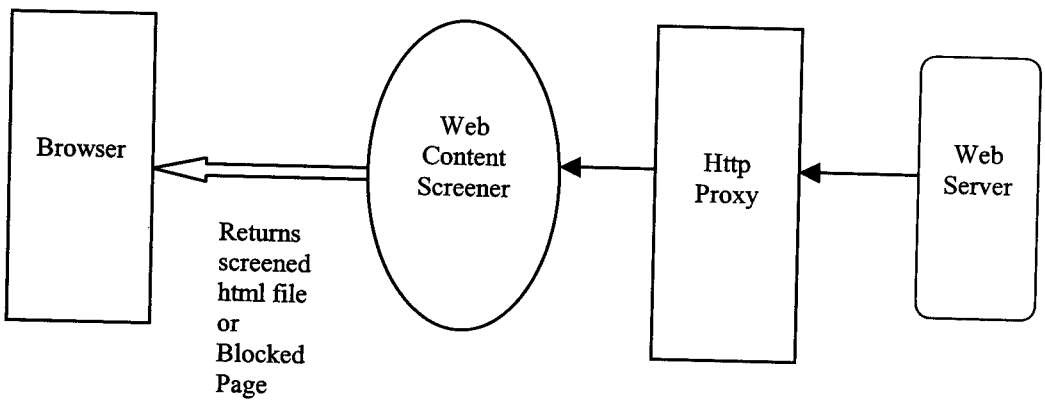
2. Content Parsing:

Once the web page is returned by the proxy server, the contents are subjected to comprehensive filtering mechanisms which search for keywords, images, advertisements, banners, popup windows & cookies. If necessary they are filtered.



3. Returning screened Page:

The filtered page or the custom page is sent back to the web browser.



IMPLEMENTATION DETAILS

IMPLEMENTATION DETAILS

We have implemented the **'WEB CONTENT SCREENER'** software in LINUX environment. The following features have been implemented in this application.

- Url Blocking
- Keyword Blocking
- Ads and Banners Blocking
- Cookies Blocking
- Popup Window Blocking
- Image Blocking

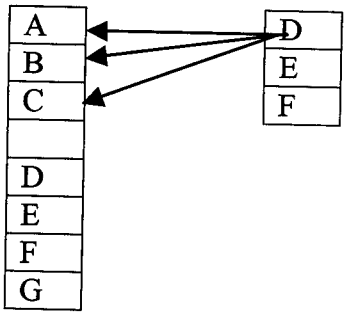
We maintain flags for each feature and store its values in a configuration file called **'firewall.con'**. When we start the application this file is read and according to the flag values screening is performed. All the features listed above need some form of word searching .So we have implemented an efficient string search algorithm called **'Boyer-Moore'** algorithm and use it wherever required. The modules keyword blocking, ads and banners blocking and url blocking make use of an access list each, which contains the strings to be searched for. We can add or remove strings from the access list, display them as and when required. Each feature has been implemented as a separate module and finally they are integrated into the application. Now let us see the implementation of the word search algorithm followed by the implementation of each feature.

Word Search Algorithm:

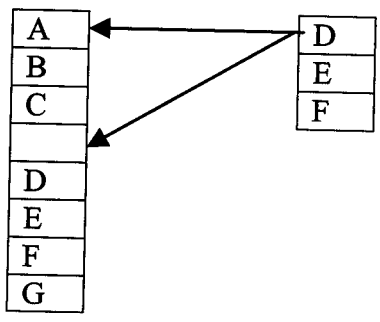
The below shown figure represents the obvious method involved in a normal string searching routine. The technique involved here is that the first character of the search string is matched with every character in the buffer until a match is found and when it is found the rest of the characters are compared in succession until the length of the search string. The disadvantage with this method as seen from the

figure is that whenever a non-match is found we can only skip one character at a time to start the next

Comparison and this is effectively solved in the next method



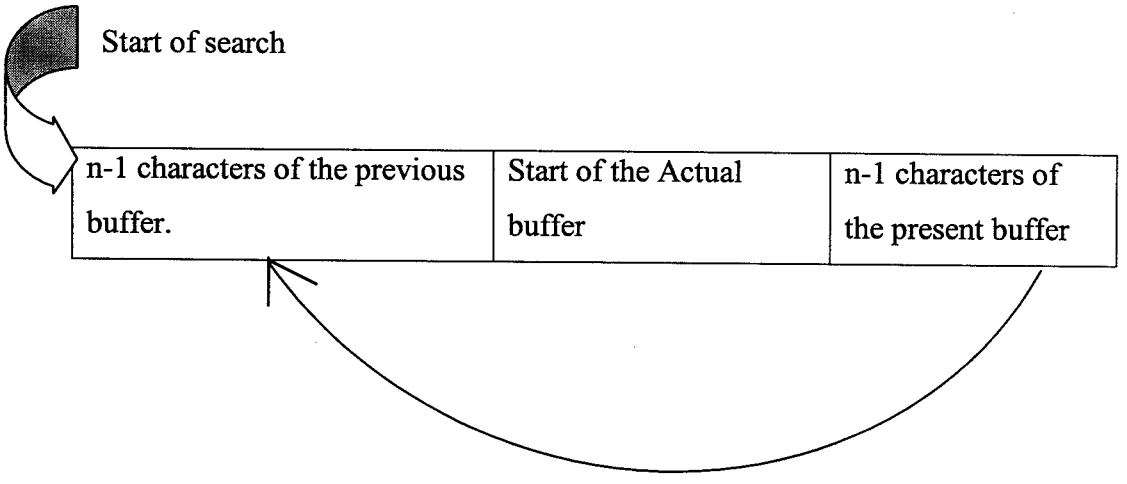
The next technique is called Boyer Moore's String searching algorithm named after its author wherein instead of checking for a match of the first character we search for a match of the last character because of which we get a chance of skipping the search string's length ahead to start the next comparison .This is shown in the following figure, at first the last character 'F' in this case is searched with it's counterpart in the buffer and as it is a mismatch and we know that there cannot be any overlapping matches ,we now skip three characters ahead and go to 'E' in the buffer instead of just skipping one as before.



Inter-block search:

There is a hidden problem in both the methods, a situation where the String to be found is partially located in the buffer already in the memory and the rest resides in the portion to be loaded next in to the memory .This problem can be solved by copying the last n-1 characters from the buffer to starting of the buffer itself. The next block is now appended to these n-1 characters in the buffer and thus the partial

characters are not lost in the subsequent searches. Here 'n' is the length of the search string.



Even in the worst case this algorithm proves to be as good as the normal sequential string search algorithm. Thus the searching part has been effectively optimized.

Features:

4.1 Url Blocking:

When a request is received from the client browser it is checked against the list of URLs got from the **'url.al'** access list that has been supplied by the user to be blocked. If a match is found in the request then a default web page is sent to the browser stating that the requested URL has been blocked by the user. If no match is found then the request is sent to the proxy server for further processing.

The user has provision to add new URLs that are to be blocked or can delete an already existing URL from the access list or just view the URLs in the access list for reference. The user also has provision to enable or disable the URL blocking module.

4.2 Keyword blocking:

Once the web page is received from the proxy server it is written into a file. This file is then searched for all the words present in the access list file **'word.al'**. We read the words from the access list one by one and then the word is searched in the file. This module makes use of the Boyer-Moore search algorithm. If a match occurs then the custom page which says that the web page has been blocked is sent. If none of the words available in the access list are found in the file, the fetched page is sent to the browser and the current connection is closed.

The user has provision to add new words that are to be blocked or can delete an already existing word from the access list or just view the words in the access list for reference. The user also has provision to enable or disable the word blocking module.

4.3 Ads and Banners Blocking:

The web page that has been stored in the file is searched for any advertisement or banner extension. The user defined extensions are stored in the access list called **'ads.al'.** First all the **'img'** tags in the file are fetched and then in each img tag the extensions available in the access list are searched . If there is a match it is replaced by the string **'ABLOCKED'** and the modified file is sent to the browser. If none of the strings match, the original html file is sent to the browser.

The user has provision to add new ad /banner extensions that are to be blocked or can delete an already existing ad/banner extension from the access list or just view the add/banner extensions in the access list for reference. The user also has provision to enable or disable the ad/banner blocking module.

4.4 Cookie Blocking:

Cookies are a general mechanism which server side connections can use to both store and retrieve information on the client side of the connection. They take up space in the hard disk. Hence we provide cookie blocking feature.This is implemented as follows. The web page that has been stored in a file is searched for the header called Set-Cookie. This header is usually sent to the browser to set a cookie. It contains the following fields.

a) Name

b) Path

d) Domain

e) Expires

e) Secure

When this header is deleted , the web page does not store these values thus blocking the cookie.There is provision to enable/disable this feature.

4.5 Popup Window Blocking:

Popup windows are additional windows that come along with a web page in order to display advertisements. For some users it is annoying. They are also a waste of system resources as the web page takes time to get loaded. To avoid these discomforts, the user can block popup windows by enabling the popup block flag. When this flag is set the web page is searched for any popup window scripts. Popup window is displayed by using a function called **'window.open'** or **'open'** function available in java script. If it is found it is removed and the resultant page is sent to the browser, else the original page is sent to the subsequent modules for further processing. The user has provision to enable or disable the popup window block module by setting a flag in the configuration file.

4.6 Image Blocking:

The web pages that we see these days contain lot of images. Images provide more information than the textual contents. But they consume lot of bandwidth and it takes time to fetch the images to get it displayed. So those users who just want the textual contents to be displayed on a page can disable the images thus saving lot of time and bandwidth.

This module searches for any **'img'** tag in the web page which are replaced by a special tag **'BLOCKED'** which disables all the subsequent image requests from the browser. Thus no images are displayed. The user also has the provision to enable or disable this module by setting a flag for image blocking in the configuration file.

TESTING

TESTING

The features that have been implemented in the software are independantly tested and then integrated into the application

5.1 Unit Testing:

Keyword Blocking:

This module is tested by sending a sample html file which contains the word in the access list as input to this function. If the word search succeeds, the custom page is sent to the browser which is verified by seeing the contents of the browser. In the event of a failure the original page is sent back to the browser which is verified by observing the original page being fetched in the browser.

Ads and Banners Blocking:

This module is tested by sending a sample html file as input to this function which contains the ad/banner extension in the **'img'** tag which matches the link in the access list. Verification is done by observing the output in the browser which contains the string **'ABLOCKED'** in the place of the extension in the html page.

Popup Window Blocking:

This module is tested by supplying a html file with a popup window as input. An output file is generated without the popup window which is checked by opening the html file in the browser. Verification is done by observing that no popup window is displayed now.

Cookie Blocking:

First we fetch a page using direct internet connection and see whether a cookie has reached the browser. Now we delete the cookie. Now we again fetch the same page using our application with the cookie block flag set. Verification is done by observing that the same cookie entry is not found again.

Image Blocking:

This module is tested by sending a sample html file which contains the **'img'** tag as input to this function. Verification is done by observing the output in the browser which contains the string **'BLOCKED'** in the place of all the img tags.

Url Blocking:

To test this module we fetch a page whose URL matches that available in the access list. If the browser has now received the custom block page it means that the url blocking feature has been verified.

5.2 Integration Testing:

After testing the modules individually we integrated them into the application and then we conducted a series of tests to verify the correctness of the application. Thus we ended up in a flawless application.

FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS

The following enhancements are possible to the following application.

- Adding a GUI to the existing application to make it more user friendly.
- Adding email spam blocking feature.
- Multiple Operating system support.
- Adding MIME type blocking feature.

CONCLUSION

CONCLUSION

Thus the **'WEB CONTENT SCREENER'** application helps the user control web access policies tailored to his/her needs. Though it requires the user to have a basic knowledge of Linux it has an easy user defined command prompt interface. We also provide a help file to the user in case he/she wants to know how to use the application and to configure it.

This application finds use in web browsing centres where they want to restrict the users from viewing unethical sites. It can also be used in educational institutes, home etc. Thus the objective of developing a HTTP gateway which can be customized according to the needs of the user to block/allow web pages has been successfully achieved.

REFERENCES

REFERENCES

- Richard W Stevens, "TCP/IP Illustrated", **Volume III**, Prentice Hall of India Private Limited, March 2003, Fourth Edition.
- Andrew S Tanenbaum, "**Computer Networks**", Pearson Education, Inc., 2003 Fourth Edition.
- Richard W Stevens, "**UNIX Network Programming**", Prentice Hall of India Private Limited, October 2002, Second Edition.
- Robert L Ziegler, "**Linux Firewalls**", Techmedia Publications, 2000, First Edition.
- Jeff Frentzen & Henry Sobotka, **"Java Script Annotated Archives"**, Tata Mc-GrawHill publication, 1999 Edition.
- Dennis M Ritchie, **"The C Programming Language",** Prentice Hall of India Private Limited, June 1999, Second Edition.
- **www.cookiecenter.com**
- **www.squid.org**

APPENDIX

APPENDIX

9.1 Sample Code:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include "algo.c"
#define MAXDATASIZE 4096

char * extracturl(char *block)
{
char *url;
url=strtok(block," ");
url=strtok(NULL," ");
return url;
}

void sendcustompage(int fd,char *filename)
{
   FILE *fp;
   char c;
   if((fp=fopen(filename,"r"))==NULL)
   {
         printf("Error Opening file");
         exit(1);
   }
   while((c=fgetc(fp))!=EOF)
         send(fd,&c,1,0);
   fclose(fp);
   shutdown(fd,SHUT_RDWR);
}

   int url_block(int fd,char *block,int bsize,char *pattern,int psize,int option)
   {

   FILE *fp;
   char wordinfile[80];
   char filename[30];
   int status=0;
   unsigned char *k=" ";
         if(option==1)
         strcpy(filename,"url.al");
         else
         strcpy(filename,"banner.al");
  if((fp=fopen(filename,"r"))==NULL)
   {
  printf("File creation error\n");
```

```
        }
        while(fgets(wordinfile,80,fp)!=NULL)
        {
                k=findstring(pattern,psize,wordinfile,(strlen(wordinfile)-1));
                printf("k=%s\n",k);
                if(k!=NULL)
                {
                status=1;
                sendcustompage(fd,"return.txt");
                }
                else
                continue;
        }
fclose(fp);
return (status);
}

int popup_block(char *filename)
{
        FILE *fp,*tmp;
        char *block,*pattern="open(",*var,c,*test,*command=(char *)malloc(40);
        unsigned int bsize,psize,pos=0;
        int pattern_count,filesizeread=0,length,status=0,count=0;
        fp=fopen(filename,"r");
        tmp=fopen("popup.htm","w");
        //strcpy(pattern," is ");

        printf("\n\n Entered Popup Block\n\n");
        length=strlen(pattern);
        pattern_count=length;
        test=(char *)malloc(150);
        var=(char*)malloc(length+1);
        block=(char *)malloc(MAXDATASIZE+length);
        memset(block,0,length);
        do
        {
                count=0;
                filesizeread=0;
                while((c=fgetc(fp))!=EOF)
                {
                        *(block+filesizeread+length)=c;
                        if(filesizeread<=(MAXDATASIZE-1))
                                filesizeread++;
                        else
                                break;
                }
                if((test=findstring(block,filesizeread+length,pattern,length))!=NULL)
                {
                        while(*(test+pos)!=')')
                        {
```

```
                        *(test+pos)=' ';
                        pos++;
                }*(test+pos)=' ';
                printf("\n\npopup window Blocked\n\n");
                pos=0;
        }
        while(count<=filesizeread)
        {
                fputc(*(block+count+length),tmp);
                count++;
        }
        while(pattern_count)
        {
                var[length-pattern_count]=block[filesizeread+length-
                pattern_count];
                pattern_count--;
        }
        *(var+1)='\0';
        strcpy(block,var);
    }while(filesizeread==MAXDATASIZE);
    fclose(fp);
    fclose(tmp);
    sprintf(command,"mv popup.htm %s",filename);
    system(command);
    remove("popup.htm");
    printf("tmp removed\n");
    return 1;
}

int word_block(char *accesslist,char *html,int new_fd)
{
    FILE *p1;char *wordinfile;
    p1=fopen(accesslist,"r");
    int status=0;
    //while(fgets(wordinfile,80,p1)!=NULL)
    while(fscanf(p1,"%s",wordinfile)!=EOF)
    {
    printf("word :%s\n",wordinfile);
    status=wordsearch(html,wordinfile);

        if(status==1)
        {
        printf("status=1 inside word block");
        sendcustompage(new_fd,"return.txt");
        break;
        }
        else continue;
    }
fclose(p1);
return status;
```

```
}
int wordsearch(char *filename,char *searchstring)
{
        FILE *fp;
        char *block,*pattern=searchstring,*var,c,*test;
        unsigned int bsize,psize;
        int pattern_count,filesizeread=0,length,status=0,count=0;
        fp=fopen(filename,"r");
        length=(strlen(pattern));
        pattern_count=length;
        var=(char*)malloc(length+1);
        block=(char *)malloc(MAXDATASIZE+length);
        memset(block,0,length);
        do
        {
                filesizeread=0;
                while((c=fgetc(fp))!=EOF)
                {
                        *(block+filesizeread+length)=c;
                        if(filesizeread<=(MAXDATASIZE-1))
                                filesizeread++;
                        else
                                break;
                }
                printf("%d\n %d\n",++count,filesizeread);
                if((test=findstring(block,filesizeread+length,pattern,length))==NULL);
                else
                {
                        status=1;
                        printf("\n\n Word Found\n\n");
                        break;
                }
                while(pattern_count)
                {
                        var[length-pattern_count]=block[filesizeread+length-
                        pattern_count];
                        pattern_count--;
                }
                *(var+1)='\0';
                strcpy(block,var);
        }while(filesizeread==MAXDATASIZE);
        printf("status=1 after word search\n");
        free(block);
        free(var);
        return status;
}

int image_block(char *filename)
{
        FILE *fp,*tmp;
```

```
Char *block,*pattern="<img",*pattern1="<IMG",*var,c;
Char *test,*replace="[BLOCKED]<AD ",*command;
int pos=0;
unsigned int bsize,psize;
int pattern_count,filesizeread=0,length,status=0,count=0;
fp=fopen(filename,"r");
tmp=fopen("image.htm","w");
printf("\n\n\n\nENtered Image Blocking\n\n\n");
length=strlen(pattern);
pattern_count=length;
test=(char *)malloc(150);
var=(char*)malloc(length+1);
block=(char *)malloc(MAXDATASIZE+length);
memset(block,0,length);
do
{
        count=0;
        filesizeread=0;
        while((c=fgetc(fp))!=EOF)
        {
                *(block+filesizeread+length)=c;
                if(filesizeread<=(MAXDATASIZE-1))
                        filesizeread++;
                else
                        break;
        }

    while((test=findstring(block,filesizeread+length,pattern,length))!=NULL ||
(test=findstring(block,filesizeread+length,pattern1,length))!=NULL)
        {
                while(*(replace+pos)!=' ')
                {
                        *(test+pos)=*(replace+pos);
                        pos++;
                }*(test+pos)=' ';
                pos=0;
        }
        while(count<=filesizeread)
        {
                fputc(*(block+count+length),tmp);
                count++;
        }
        while(pattern_count)
        {
                var[length-pattern_count]=block[filesizeread+length-
                pattern_count];
                pattern_count--;
        }
        *(var+1)='\0';
        strcpy(block,var);
```

```
        }while(filesizeread==MAXDATASIZE);
        fcloseall();
        sprintf(command,"mv image.htm %s",filename);
        system(command);
        remove("image.htm");
        return 1;
}
void cookie_block(char *filename)
{
FILE *fp,*fp1;
char *pattern="Set-Cookie",*command=(char *)malloc(20);
char line[80];
int bsize;
fp=fopen(filename,"r");
if((fp1=fopen("cookie.htm","w"))==NULL)
{
printf("could not create file\n");
return;
}

        printf("block entered\n");
        while(fgets(line,80,fp)!=NULL)
        {
                if(findstring(line,strlen(line),pattern,10)!=NULL)
                {
                        printf("cookie header found\n");
                        continue;
                }
                else
                        fputs(line,fp1);
        }
        fputs("\n",fp1);
        fclose(fp);
        fclose(fp1);
sprintf(command,"mv cookie.htm %s",filename);
system(command);
remove("cookie.htm");
}
int ad_block(char *filename,char *accesslist)
{
        FILE *fp,*p1,*p2;
        char *block,*pattern="<img ",*pattern1="<IMG
        ",c,*replace="[ABLOCKED]<AD ",*adword=(char *)malloc(60),*command;
        char *test=(char *)malloc(150);
        int bsize=0,pos=0,pos1=0,count=0,i=0;
        char *ad=(char *)malloc(150);
        char *tmp;
                if((fp=fopen(filename,"r"))==NULL)
                {
                        printf("could not open mod file\n");
```

```
}
if((p1=fopen(accesslist,"r"))==NULL)
{
        printf("could not open accesslist\n");
}
if((p2=fopen("ad.htm","a+"))==NULL)
{
        printf("could not open adtempdir\n");
}
while(fgetc(fp)!=EOF)
{
        bsize++;
}
rewind(fp);
if((block=(char *)malloc(bsize))==NULL)
{
        printf("could not allocate memory\n");
}
while((c=fgetc(fp))!=EOF)
{
        *(block+i)=c;
        i++;
}
*(block+i)='\0';
while(((test=findstring(block,bsize,pattern,5))!=NULL) ||
(test=findstring(block,bsize,pattern1,5))!=NULL)
{
        while(*(test+pos)!='>')
        {
                *(ad+pos)=*(test+pos);
                pos++;
        }
        *(ad+pos)='>';
        *(ad+pos+1)='\0';

                while(fscanf(p1,"%s",adword)!=EOF)
                {

if(findstring(ad,pos+1,adword,strlen(adword))!=NULL)
                        {
                                while(*(replace+pos1)!=' ')
                                {
                                        *(test+pos1)=*(replace+pos1);
                                        pos1++;
                                }
                        pos1=0;
                        break;
                        }
                }
                for(count=0;count<((test+pos)-block);count++)
```

```
                        {
                                fputc(*(block+count),p2);
                        }
                        rewind(p1);
                memset(ad,0,150);
                bsize=bsize-(test+pos+1-block);
                printf("%d\n",test+pos-block);
                block=test+pos+1;
                tmp=test;
                pos=0;
        }
        while(*tmp)
        {
                fputc(*tmp++,p2);
        }
        free(block);
        free(test);
    fclose(fp);
    fclose(p1);
    fclose(p2);
    sprintf(command,"mv ad.htm %s",filename);
    system(command);
    remove("ad.htm");
return 0;
}
```

9.2 Sample Output:

Custom Block Page:

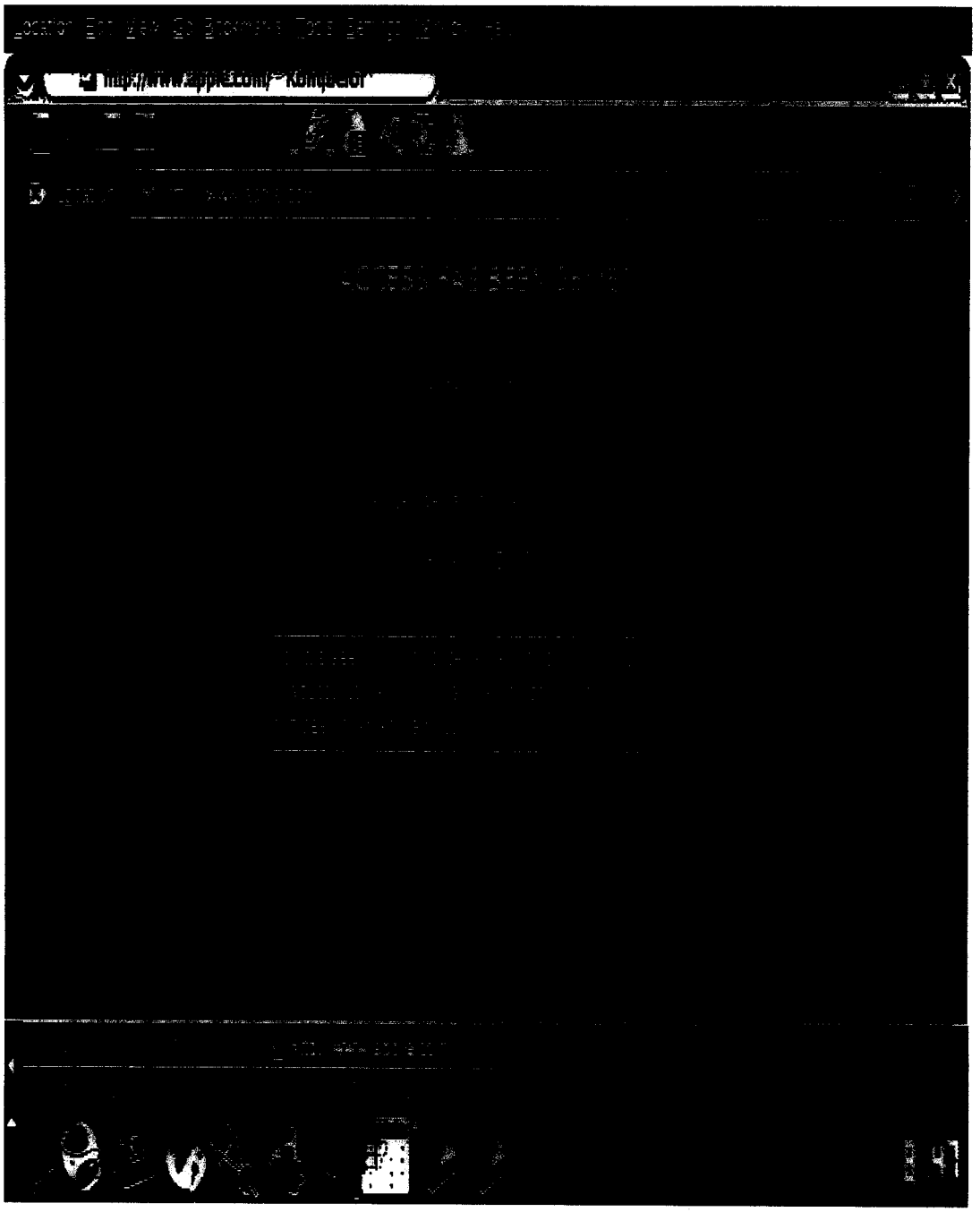


Image Block Page: