

NetNinny_JM

Compiling:

Extract the package contents into a folder. Go to that folder and run 'make' in terminal.

Running:

`./NetNinny_JM [port number]` e.g.: `./NetNinny_JM 51717`

Change proxy settings in the browser:

Proxy IP address: 127.0.0.1

Port: same as you gave as a parameter to NetNinny_JM

Go to a web page.

Features:

1. Handles simple HTTP GET interactions between client and server -
main.cpp, lines 66-78
2. Blocks requests for undesirable URLs, using HTTP redirection to display an error page instead
- main.cpp lines 82, 155-158
3. Detects inappropriate content bytes within a Web page before it is returned to the user, and
redirecting to an error page -
main.cpp lines 130-152
4. Imposes no limit on the size of the transferred HTTP data - uses realloc() to put together the
response. Buffer should expand until it reaches operating system limits, which rarely happens. -
appendToBuffer, in main.cpp, line 116
5. Is compatible with all major browsers (e.g. Internet Explorer, Mozilla Firefox, Google Chrome,
etc.) without the requirement to tweak any advanced feature
- tested only with Firefox 10.2.0 on Linux Mint
6. Allows the user to select the proxy port (i.e. the port number should not be hard coded) -
port number given as a parameter - main.cpp, line 42
7. Is smart in selection of what HTTP content should be searched for the forbidden keywords. For
example, you probably agree that it is not wise to search inside compressed or other non-text-
based HTTP content such as graphic files, etc. -
main.cpp line 130 - hasPlainContent checks whether the response contains the word "text" in its
header. If it returns true, content is searched for keywords.

Limitations:

HTTP request can be up to 4096 bytes long. This is enough for almost all requests, unless the cookie is large or you're uploading a file/form with POST.

Also, the proxy crashes when there are a lot of components which have to be fetched from different servers. And it's unable to retrieve favicon, because getaddrinfo() function interprets favicon.com as a hostname and is unable to retrieve an IP address for it.

Testing:

Testing was done with Firefox 10.2.0 on a computer running Linux Mint 10. Connected through university's network, eduroam either LiU.

Some URLs with which the proxy was tested:

works well:

<http://www.gnu.org/software/make/>

<http://computer-graphics.se/gamejam/>

<http://www.infosa.lt/main.php>

<http://azuolas.org/main/lt/news/>

doesn't work:

<http://listen.grooveshark.com/#/> - answers with "400 Bad request"

<http://en.wikipedia.org/> - says "server error", although works without the proxy

<http://yahoo.com/> - loads partly, but the proxy crashes when trying to fetch page components from other hostnames (gfx.svd-cdn.se in this case)

<http://news.ycombinator.com/news> - answers with the word "unknown"

<http://www.student.liu.se/?l=en> - loads most of the components, but in the end throws:

```
NetNinny_JM: malloc.c:3096: sYSMALLOc: Assertion `(old_top == (((mbinptr) (((char *) &((av)->bins[((1) - 1) * 2])) - __builtin_offsetof (struct malloc_chunk, fd)))) && old_size == 0) || ((unsigned long) (old_size) >= (unsigned long)(((__builtin_offsetof (struct malloc_chunk, fd_nextsize))+((2 * (sizeof(size_t))) - 1)) & ~((2 * (sizeof(size_t))) - 1))) && ((old_top)->size & 0x1) && ((unsigned long)old_end & pagemask) == 0)' failed.
```

Aborted

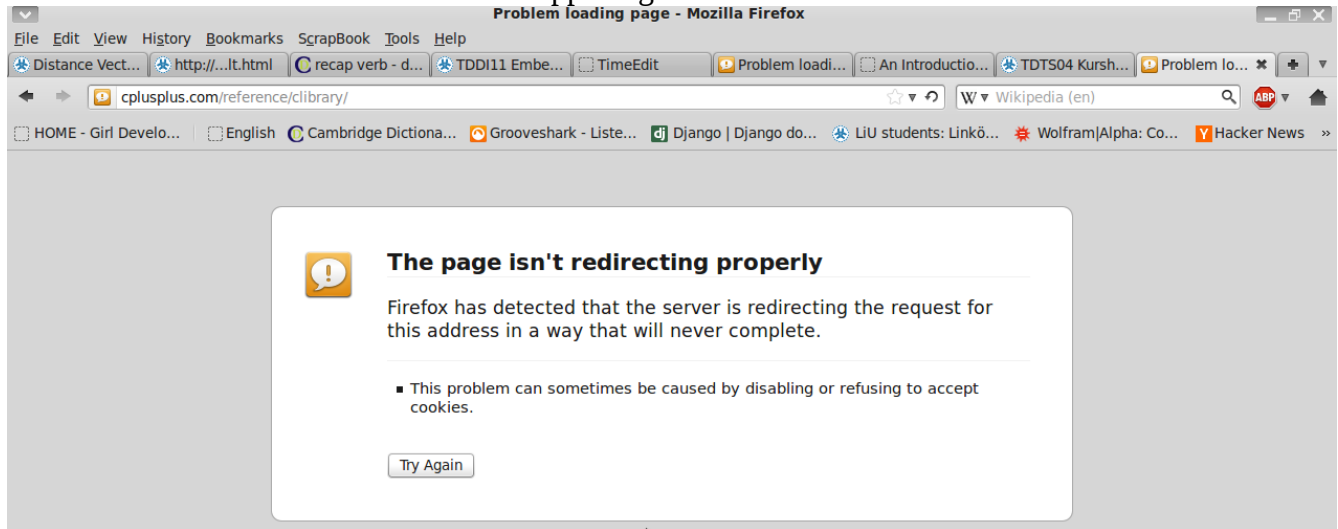
<http://translate.google.com/> - loads, but does not translate a word

<http://stackoverflow.com/> - responds with "HTTP Error 400. The request is badly formed."

Below there are a few responses from pages which did not load using the proxy.

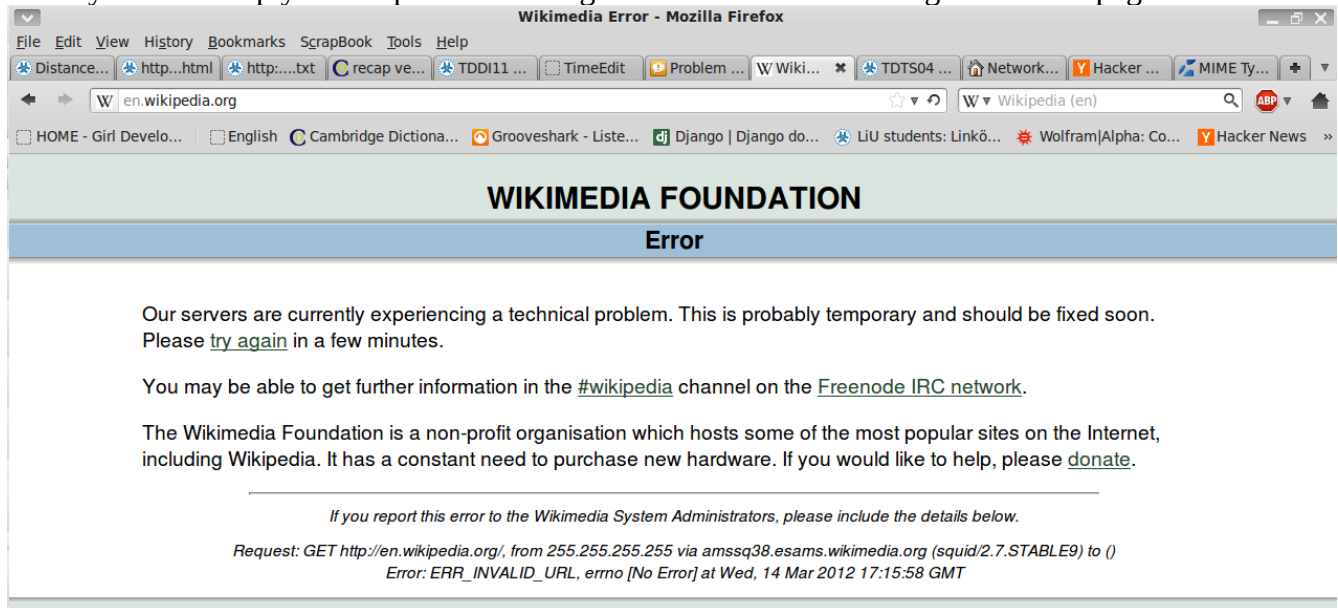
cplusplus.com:

the browser detected that some mess is happening with redirection



<http://en.wikipedia.org/>:

This may have been because few sites still use non-persistent connections, because it's less effective. So they refuse to reply to a request containing "Connection: close" and give an error page instead.



Code:

main.cpp:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

#include "utilities.h"

int main(int argc, char *argv[])
{
    // variables
    int sockfd1C,    // client-listening socket
    sockfd2C,        // client-proxy communication socket
    sockfd1S,        // proxy-host server communication socket
    portnoC, status;

    socklen_t clilen;
    char *keywords, *redirectReq;
    struct sockaddr_in serv_addr, cli_addr;

    // error check
    if (argc < 2) {
        fprintf(stderr, "ERROR, no port provided\n");
        exit(1);
    }
    sockfd1C = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd1C < 0)
        error("ERROR opening socket");

    // read keywords and redirect request from files
    keywords = readWords();
    redirectReq = getRedRequest();

    // setup and bind a socket that listens to client requests
    bzero((char *) &serv_addr, sizeof(serv_addr)); // fill serv_addr with zeros
    portnoC = atoi(argv[1]); // turn string to number
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portnoC); // host to network short
    if (bind(sockfd1C, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR on binding");

    bool quit = false;

    while(!quit)
    {
```

```

        char *buffer = NULL, *recv_buffer = NULL, // request buffer, response
buffer
        *hostname;

        if( !(buffer = (char*) malloc(cBufSize)) ) printf("Failed to allocate
request buffer\n");

        int req_sz, resp_sz = 0;
        // char ipstr[INET6_ADDRSTRLEN];

        printf("-----\nThe proxy is ready\n");

        // gets request from browser
        //-----
-----
        listen(sockfd1C, 5); // 5 - max # of pending connections

        clilen = sizeof(cli_addr);
        // accepts client connection
        sockfd2C = accept(sockfd1C, (struct sockaddr *) &cli_addr, &clilen);

        bzero(buffer, cBufSize);

        req_sz = recv(sockfd2C, buffer, cBufSize-1, 0); // read from socket2C

        if(DEBUG) printf(">%s<\n\n", buffer); // print out request
        //-----
        -----

        // checks URL. If ok - retrieves page, else - redirects
        if ( isAppropriate(buffer, cBufSize, keywords) ) // checks http request
for forbidden words
        {
            printf("Request is OK\n");
            if( setNonpersistent(buffer)) printf("\nConnection: close\n" set\n");
            printf(">%s<\n\n", buffer); // print out request

            hostname = findHostNm(buffer);

            printf("hostname: %s\n", hostname);

            // connects to server, returns server socket
            sockfd1S = connectToServer(hostname);

            // gets response and stores in varied-size recv_buffer
            //-----
            -----

            int bytes_read, bytes_sent = send(sockfd1S, buffer, req_sz, 0);
            char *tmp;

            if(bytes_sent > 0)
                printf("Request sent (%d bytes)\n", bytes_sent);
            else
                error("ERROR sending request");

```

```

        if( !(tmp = (char*) malloc(cBufSize)) ) printf("Failed to allocate tmp
buffer\n");

        do
        {
            bzero(tmp, cBufSize);
            bytes_read = recv(sockfd1S, tmp, cBufSize, 0);
            // printf("%d bytes received\n", bytes_read);
            // printf("\n%s", tmp);

            if(bytes_read > 0){
                recv_buffer = (char*) realloc( (void*) recv_buffer, resp_sz +
bytes_read);
                memcpy( (void*) (recv_buffer + resp_sz), tmp, bytes_read);
                resp_sz += bytes_read;
            }

            } while( bytes_read > 0 );

            if(DEBUG) printf("Response got (%d bytes)\n", resp_sz);
            //if(DEBUG) printf("%s\n", recv_buffer);
            //-----
            -----

            // checks the content and either forwards to the browser or redirects
            //-----
            -----

            if( hasPlainContent(recv_buffer) ){
                if(DEBUG) printf("Contains text -> check:\n");
                if( isAppropriate(recv_buffer, resp_sz, keywords) )
                {
                    if( (status = send(sockfd2C, recv_buffer, resp_sz, 0) < 0))
                        fprintf(stderr, "Passing response to client failed: %s\n",
gai_strerror(status));
                    else
                        printf("Response sent to
client\n-----\n\n");

                }
                else
                {
                    printf("Redirecting\n");
                    redirect(sockfd2C, redirectReq);
                }
            }
            else
            {
                if( (status = send(sockfd2C, recv_buffer, resp_sz, 0) < 0))
                    fprintf(stderr, "Passing response to client failed: %s\n",
gai_strerror(status));
                else
                    printf("Response sent to
client\n-----\n\n");
            }
            //-----
            -----

```

```

        } else {
            printf("Forbidden words found in URL!\nRedirecting\n");
            redirect(sockfd2C, redirectReq);
        }

        free(buffer);
        free(recv_buffer);
    }

    close(sockfd2C);
    close(sockfd1C);
    close(sockfd1S);

    free(keywords);
    free(redirectReq);

    return 0;
}

```

utilities.cpp:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include "utilities.h"

const char* cKWFile = "words";
const char* cRedReqFile = "redirect";
const int cWords = 256;

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

void toLowercase(char *string)
{
    int i = 0;
    while(string[i] != '\0')
    {
        if( string[i] >= 65 && string[i] <= 90)
        {
            string[i] += 32;
        }
        i++;
    }
}

```

```

bool isAppropriate(char *buffer0, int buf_sz, char* kWords)
{
    char *word, *buffer,
        *keywords = (char*) malloc(cWords);

    buffer = (char*) malloc(buf_sz);
    buffer = strcpy(buffer, buffer0);
    keywords = strcpy(keywords, kWords);
    toLowercase(buffer);

    //printf("keywords:\n%s\n", keywords);

    word = strtok (keywords, ",");
    while (word != NULL)
    {
        toLowercase(word);
        printf("checking: %s\n", word);
        if ( strstr(buffer, word) == NULL )
        {
            word = strtok (NULL, ",");
        } else {
            free(buffer);
            return 0;
        }
    }
    //printf("\n%s\n", kWords);
    free(buffer);
    return 1;
}

```

```

char* readWords()
{
    FILE * file;
    file = fopen(cKWFile, "r");

    char *keywords = (char*) malloc(cWords);

    if( !fread(keywords, 1, cWords, file) )
    {
        printf("Failed to read words from file\n");
        return NULL;
    }
    // printf("The forbidden words:\n%s< end.\n", keywords);

    fclose(file);
    printf("keywords loaded\n");
    return keywords;
}

```

```

char* getRedRequest()
{
    FILE * file;
    file = fopen(cRedReqFile, "r");
    int bytes;
    char *redirectReq = (char*) malloc(cBufSize);
    if( (bytes = fread(redirectReq, 1, cBufSize, file)) <= 0 )
    {

```



```

        printf("Failed to read redirectReq from file\n");
        return NULL;
    }
    printf("redirectReq loaded, %d bytes\n", bytes);
    fclose(file);
    return redirectReq;
}

char* findHostNm(char buffer[])
{
    char *hostNm = (char*) malloc(256), *buffer1;
    if ( strstr(buffer, "Host: ") != NULL )
    {
        buffer1 = ( strstr(buffer, "Host: ") + 6);
        hostNm = strtok(buffer1, "\n\r");
        // sprintf(hostNm, "%s\0", strtok(buffer1, "\n"));

    }
    return hostNm;
}

char* setNonpersistent(char *buffer)
{
    char *start;

    start = strstr(buffer, "Proxy-Connection: keep-alive");
    if ( start != NULL )
    {
        buffer = ( strncpy(start, "Connection: close", 28));
    }

    return start;
}

bool hasPlainContent(char *buffer)
{
    char *start;
    if(! (start = strstr(buffer, "Content-Type")))
        start = strstr(buffer, "content-type");

    if(start && strstr(start, "text"))
        return true;
    else
        return false;
}

int retrievePage(int sockfd1S, char* buffer, char **recv_buffer, int req_sz)
{
    int bytes_read, bytes_sent = send(sockfd1S, buffer, req_sz, 0),
        resp_sz = 0;
    char *tmp;

    if(bytes_sent > 0)
        printf("Request sent (%d bytes)\n", bytes_sent);
}

```

```

        else
            error("ERROR sending request");

        if( !(tmp = (char*) malloc(cBufSize)) ) printf("Failed to allocate tmp
buffer\n");

        do
        {
            bzero(tmp, cBufSize);
            bytes_read = recv(sockfd1S, tmp, cBufSize, 0);
            printf("%d bytes received\n", bytes_read);
            printf("\n%s", tmp);

            if(bytes_read > 0){
                *recv_buffer = (char*) realloc( (void*) *recv_buffer, resp_sz +
bytes_read);
                memcpy( (void*) ((*recv_buffer) + resp_sz), tmp, bytes_read);
                resp_sz += bytes_read;
            }

        } while( bytes_read > 0 );

        // printf("got %d bytes\n%s", resp_sz, *recv_buffer);
        return resp_sz;
    }

```

```

void redirect(int sockfd2C, char *redirectReq)
{
    char *recv_buf = NULL;

    int sockfd1S = connectToServer("www.ida.liu.se");

    int resp_sz = retrievePage(sockfd1S, redirectReq, &recv_buf, redReq_sz);
    if( send(sockfd2C, recv_buf, resp_sz, 0) == 0)
        error("Redirecting failed");
    else
        printf("Redirected successfully\n");
}

```

```

int connectToServer(char *hostname)
{
    struct addrinfo hints, *serverInfo = NULL;
    int status;

    memset(&hints, 0, sizeof(hints)); // make sure the struct is empty
    hints.ai_family = AF_UNSPEC;      // don't care IPv4 or IPv6
    hints.ai_socktype = SOCK_STREAM; // TCP stream sockets
    hints.ai_flags = AI_PASSIVE;      // fill in my IP for me

    if ((status = getaddrinfo(hostname, "80", &hints, &serverInfo)) != 0)
    {
        fprintf(stderr, "Helvete! It's a getaddrinfo error: %s\n",
gai_strerror(status));
        exit(1);
    }
}

```

```

    int sockfd1S = socket(serverInfo->ai_family, serverInfo->ai_socktype,
serverInfo->ai_protocol); // open socket
    if (sockfd1S < 0)
        error("ERROR opening socket");
    else
        printf("Socket open\n");

    if ( connect(sockfd1S, serverInfo->ai_addr, serverInfo->ai_addrlen) < 0)
        error("ERROR connecting");
    else
        printf("Connected\n");

    freeaddrinfo(serverInfo);

    return sockfd1S;
}
//
//    printf("\nIP addresses for %s:\n", hostname);
//    for(p = serverInfo; p != NULL; p = p->ai_next)
//    {
//        void *addr;
//        char *ipver;
//
//        // get the pointer to the address itself,
//        // different fields in IPv4 and IPv6:
//        if (p->ai_family == AF_INET) { // IPv4
//            struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
//            addr = &(ipv4->sin_addr);
//            ipver = (char*)"IPv4";
//        } else { // IPv6
//            struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p-
>ai_addr;
//            addr = &(ipv6->sin6_addr);
//            ipver = (char*)"IPv6";
//        }
//        // convert the IP to a string and print it:
//        inet_ntop(p->ai_family, addr, ipstr, sizeof(ipstr) );
//        printf("    %s: %s\n", ipver, ipstr);
//    }

```

utilities.h:

```

#ifndef UTILITIES_H
#define UTILITIES_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

```

```
const int cBufSize = 4096;
const int portnoS = 80; // HTTP port no.
const int redReq_sz = 370;

const bool DEBUG = true;

void error(const char *msg);

bool isAppropriate(char *buffer, int buf_sz, char keywords[]);
char* readWords();
char* getRedRequest();

void toLowercase(char *string);
char* findHostNm(char buffer[]);

char* setNonpersistent(char* buffer);
int retrievePage(int sockfd1S, char* buffer, char **recv_buffer, int req_sz);
void redirect(int sockfd2C, char *redirectReq);
int connectToServer(char *hostname);
bool hasPlainContent(char *buffer);

#endif
```