Keith Tunstead
Hitesh Tewari
20 February 2017

Telecommunications Proxy Server

Design and Documentation

Description

The idea of this proxy web server is that it should sit between the users browser and the outside internet. Basically, we setup a socket connection to the browser (on port 2000) to listen for incoming connections. This means that any requests made on the browser (on port 2000) are routed to the running proxy. The proxy then creates a new thread to take the received connection, along with the request header from the browser, determines the base url of the request and the sub path of the request and checks if the base url is blacklisted. If the host is not blacklisted, we continue with the request. We setup another connection, this time to the requested server and send the required GET request to it. When we get the response back from the server we can then parse the response header to verify a code 200 response (success). I did not however implement this as I wanted the proxy to have as little impact on what the browser was seeing, e.g. if the proxy got a 404 (not found), then this response would be relayed to the browser in order to maintain transparency. (Please see "connection error" for my exception to this). On success, we then check the cache to see if the response document is present in the cache. If it is present, we check if the current cache entry is outdated (by checking the Last-Modified header entry), if it's outdated then we update the cache, if not we forward the cached entry to the browser. If the response is not present in the cache, then we must save to the corresponding path in the cache and forward it on to the browser. At this point the thread is closed. If the host is blacklisted then we simply block the request.

Resources

One of the main issues I found with this project was how to handle resources on a webpage, eg image files, css files and js files etc. I decided to take the simplest option to handle this, that being to parse the incoming response body, checking for relevant resource request (eg "src= *"). If these resources are determined to be relative paths then we must route them through our proxy (localhost:2000) and add on the base host name. These resource requests will then be routed through the proxy, using the correct path to find the file and will then parse that file for any resources contained. This method gets most webpages to display correctly, with the exception of youtube and media websites in general, I believe this is due to the data transfer method of video and audio media.

Connection Error

If there is a connection error between the proxy and the end requested server, we check the cache for an entry to the requested path. If present, we forward this onto the browser. This is the only error handling done on the proxy. It would have been nice to handle 301 "Permanently moved" responses and to replace the request with a request for the items new location in the response header and also to handle 404 errors in a similar fashion, however, given the time restraints, I was unable to implement this error checking.

Management Console

The management console is very simple. It has 3 commands, "block" used to block the specified host, "unblock" used to unblock the specified host and "close" used to close the management console. The list of blocked hosts is maintained in a simple blacklist.txt file. This way, the proxy does not have to be restarted in order for any changes in the blacklist file to take effect.

Source Code

```
from dateutil import parser
import os,sys,thread,socket,errno,httplib



#********* CONSTANT VARIABLES *********
BACKLOG = 50              # how many pending connections queue will hold
MAX_DATA_RECV = 999999  # max number of bytes we receive at once
#**************************************
#********* MAIN PROGRAM ***************
#**************************************
def main():
    #Setup proxy to connect to browser

    # host and port info.
    port = 2000
    host = ''               # blank for localhost

    print "Proxy Server Running on ",host,":",port

    try:
        # create a socket
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # associate the socket to host and port
        s.bind((host, port))

        # listenning for a request from the browser
        print "Listening..."
        s.listen(BACKLOG)

    except socket.error, (value, message):
        if s:
            s.close()
        print "Could not open socket:", message
        sys.exit(1)

    # get the connection from browser
    threadCount = 0
    while 1:
        conn, client_addr = s.accept()

        threadCount = threadCount+1
        print "Connection accepted, creating thread..", threadCount


        # create a thread to handle request
        thread.start_new_thread(proxy_thread, (conn, client_addr, threadCount))


    s.close()
#************** END MAIN PROGRAM ***************

#****************************************
#********* PROXY_THREAD FUNC ************
# A thread to handle request from browser
#****************************************
def proxy_thread(conn, client_addr, threadID):

    # get the request from browser
    request = conn.recv(MAX_DATA_RECV)

    #print "Data Recevied from browser:"
    #print request

    splitMessage = request.split()

    Req_Type = splitMessage[0]
    Req_path = splitMessage[1]
    Req_path = Req_path[1:]
    #print "Request is ", Req_Type, " to URL : ", Req_path

    # find the webserver from req_path
    http_pos = Req_path.find("://")          # find pos of ://
    if (http_pos==-1):
        url = Req_path
    else:
        url = Req_path[(http_pos+3):len(Req_path)]    # get the rest of url (without '/' at
end)

    sub_pos = url.find("/")
    if (sub_pos==-1):
        subpath = "/"
        host_name = str(url)
    else:
        subpath = str(url[sub_pos:])       # get the rest of sub path
        host_name = str(url[0:sub_pos])       #remove subpath from url for host
```

```
    #print "HOST_NAME:", host_name
    #print "SUB_PATH:", subpath


    #Check if hostname is blocked
    bf = open("blacklist.txt","r")
    bLines = bf.read()
    if bLines.find(host_name) != -1:
        print host_name, "CURRENTLY BLACKLISTED"
        conn.close()
        print("Exiting thread..", threadID)
        return



    # ******* HTTPLIB ******
    useCache = False
    connErr = False
    try:
        conn2 = httplib.HTTPSConnection(host_name)

        conn2.request("GET", subpath)

        r1 = conn2.getresponse()

        #print r1.status, r1.reason

        header = "HTTP/1.0 " + str(r1.status) + " " + str(r1.reason) + "\r\n" + str(r1.msg) +
"\r\n"

        body = str(r1.read())
    except:
        print "Connection error occurred, will attempt to use any cached files available."
        useCache = True
        connErr = True


    # Get the subpath of the request
    relPath = ""
    filename = host_name
    if subpath != "/":
        #find last '/'
        l = subpath.rfind("/")
        relPath = subpath[0:l]
        #get the filename
        if subpath.find('.') != -1:
            filename = subpath[l+1:]

    #print "RELPATH:", relPath
    #print "FILENAME:", filename


    # OS File handling
    path_list = relPath.split(os.sep)

    dirname=os.path.dirname
    path = os.path.join(dirname(dirname(__file__)), "cache", host_name)
    for p in path_list:
        path = os.path.join(path,p)

    #print "PATH:", path

    #Create the directory
    try:
        os.makedirs(path)
    except OSError:
        if not os.path.isdir(path):
            raise


    if path.endswith("/"):
        fullPath = path + filename
    else:
        fullPath = path + "/" + filename
    #print "FILE:", fullPath


    # check if we should use cache entry or not
    try:
        if os.path.exists(fullPath):

            #If a network error occurred, use the cached version, if it exists
            #If an error occurred r1 may not exist
            if not connErr:
                if r1.status != 200:
                    print "Using cached version due to", r1.status, "ERR"
                    useCache = True


            #check cache file for Last-Modified
            lmCacheLine = ""
            for line in open(str(fullPath) ,'r').readlines():
                t = line.find("Last-Modified:")
                if t != -1:
                    lmCacheLine = line
```

```
                    break

            #If an error occurred header may not exist
            if not connErr:
            #check header for Last-Modified
                lmPos = header.find("Last-Modified:")

                if lmPos != -1 and lmCacheLine != "":
                    # Last modified in both cache file and new file
                    lastModStr = header[lmPos+20:lmPos+40]
                    lastMod = parser.parse(lastModStr)

                    lastModCacheStr = lmCacheLine[20:40]
                    lastModCache = parser.parse(lastModCacheStr)

                    if lastModCache.date() == lastMod.date():
                        #in date cache
                        print "Cache in date"
                        useCache = True

        else:
            print "Cache file not available"
            useCache = False
    except OSError:
        raise

    if useCache:
        # Read cached file
        file = open(str(fullPath) ,'r')
        fileData = file.read()
        print "Sending cached data to browser.."
        conn.sendall(fileData)
    elif not connErr:
        #Pull new file
        # we need to parse the body and change all relative links to absolute links
        # **** SPECIAL CASES ****
        #eg src="//assets.juicer.io/embed.js"
        body = body.replace("src=\"//", "src=\"http://localhost:2000/")
        body = body.replace("href=\"//", "href=\"http://localhost:2000/")
        # **** SPECIAL CASES ****

        # find all 'src' and 'href' and url("/ and srcset="/
        body = body.replace("src=\"/", "src=\"http://localhost:2000/" + host_name + "/")
        body = body.replace("href=\"/", "href=\"http://localhost:2000/" + host_name + "/")
        body = body.replace("srcset=\"/", "srcset=\"http://localhost:2000/" + host_name +
"/")
        body = body.replace("url(\"/", "url(\"http://localhost:2000/" + host_name + "/")
        body = body.replace("url(/", "url(http://localhost:2000/" + host_name + "/") #No
quotes
        #print body

        data = header + body

        #Create/Overwrite the file
        print "Writing to cache..", fullPath
        file = open(str(fullPath) ,'w+')
        file.write(data)
        file.close()
        print "Sending new data to browser.."
        conn.sendall(data)



    conn2.close()
    # ******* HTTPLIB ******


    conn.close()
    print("Exiting thread..", threadID)


#********** END PROXY_THREAD ***********
if __name__ == '__main__':
    main()
```