"""

Simple HTTP Live Streaming client.

References:

http://tools.ietf.org/html/draft-pantos-http-live-streaming-08

This program is free software. It comes without any warranty, to

the extent permitted by applicable law. You can redistribute it

and/or modify it under the terms of the Do What The Fuck You Want

To Public License, Version 2, as published by Sam Hocevar. See

http://sam.zoy.org/wtfpl/COPYING for more details.

Last updated: July 22, 2012

MODIFIED BY shani to make it work with F4mProxy

"""

import urlparse, urllib2, subprocess, os,traceback,cookielib,re,Queue,threading

import xml.etree.ElementTree as etree

import base64

from struct import unpack, pack

import struct

import sys

import io

import os

import time

import itertools

import xbmcaddon

import xbmc

import urllib2,urllib

import traceback

import urlparse

import posixpath

import re

import hmac

import hashlib

import binascii

import zlib

from hashlib import sha256

import cookielib

import array, random, string

import requests

#from Crypto.Cipher import AES

'''

from crypto.cipher.aes import AES

from crypto.cipher.cbc import CBC

from crypto.cipher.base import padWithPadLen

from crypto.cipher.rijndael import Rijndael

from crypto.cipher.aes\_cbc import AES\_CBC

'''

gproxy=None

gauth=None

nsplayer=False

callbackDRM=None

try:

from Crypto.Cipher import AES

USEDec=1 ## 1==crypto 2==local, local pycrypto

except:

print 'pycrypt not available using slow decryption'

USEDec=3 ## 1==crypto 2==local, local pycrypto

if USEDec==1:

#from Crypto.Cipher import AES

print 'using pycrypto'

elif USEDec==2:

from decrypter import AESDecrypter

AES=AESDecrypter()

else:

from f4mUtils import python\_aes

#from decrypter import AESDecrypter

iv=None

key=None

value\_unsafe = '%+&;#'

VALUE\_SAFE = ''.join(chr(c) for c in range(33, 127)

if chr(c) not in value\_unsafe)

SUPPORTED\_VERSION=3

cookieJar=cookielib.LWPCookieJar()

clientHeader=None

class HLSRedirector():

global cookieJar

"""

A downloader for f4m manifests or AdobeHDS.

"""

def \_\_init\_\_(self):

self.init\_done=False

def sendVideoPart(self,URL, file, chunk\_size=4096):

for chunk in download\_chunks(URL):

send\_back(chunk,file)

return

def init(self, out\_stream, url, proxy=None,use\_proxy\_for\_chunks=True,g\_stopEvent=None, maxbitrate=0, auth='', callbackpath="", callbackparam=""):

global clientHeader,gproxy,gauth

try:

self.init\_done=False

self.init\_url=url

clientHeader=None

self.status='init'

self.proxy = proxy

self.auth=auth

self.callbackpath=callbackpath

self.callbackparam=callbackparam

if self.auth ==None or self.auth =='None' or self.auth=='':

self.auth=None

if self.auth:

gauth=self.auth

if self.proxy and len(self.proxy)==0:

self.proxy=None

gproxy=self.proxy

self.use\_proxy\_for\_chunks=use\_proxy\_for\_chunks

#self.out\_stream=out\_stream

if g\_stopEvent: g\_stopEvent.clear()

self.g\_stopEvent=g\_stopEvent

self.maxbitrate=maxbitrate

if '|' in url:

sp = url.split('|')

url = sp[0]

clientHeader = sp[1]

print clientHeader

clientHeader= urlparse.parse\_qsl(clientHeader)

print 'header recieved now url and headers are',url, clientHeader

self.status='init done'

self.url=url

return True# disabled downloadInternal(self.url,None,self.maxbitrate,self.g\_stopEvent , self.callbackpath, self.callbackparam, testing=True)

except:

traceback.print\_exc()

self.status='finished'

return False

def keep\_sending\_video(self,dest\_stream, segmentToStart=None, totalSegmentToSend=0):

try:

self.status='download Starting'

downloadInternal(self.url,dest\_stream,self.maxbitrate,self.g\_stopEvent , self.callbackpath, self.callbackparam)

except:

traceback.print\_exc()

print 'setting finished'

self.status='finished'

def getUrl(url,timeout=15,returnres=False,stream =False):

global cookieJar

global clientHeader

global nsplayer

try:

post=None

print 'url',url

session = requests.Session()

session.cookies = cookieJar

headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux i686; rv:42.0) Gecko/20100101 Firefox/42.0 Iceweasel/42.0'}

if clientHeader:

for n,v in clientHeader:

headers[n]=v

if nsplayer:

print 'nsplayer is true'

headers['User-Agent']=binascii.b2a\_hex(os.urandom(20))[:32]

print 'nsplayer', nsplayer,headers

proxies={}

if gproxy:

proxies= {"http": "http://"+gproxy}

#import random

#headers['User-Agent'] =headers['User-Agent'] + str(int(random.random()\*100000))

if post:

req = session.post(url, headers = headers, data= post, proxies=proxies,verify=False,timeout=timeout,stream=stream)

else:

req = session.get(url, headers=headers,proxies=proxies,verify=False ,timeout=timeout,stream=stream)

req.raise\_for\_status()

if returnres:

return req

else:

return req.text

except:

print 'Error in getUrl'

traceback.print\_exc()

raise

return None

def download\_chunks(URL, chunk\_size=4096, enc=None):

#conn=urllib2.urlopen(URL)

#print 'starting download'

conn=getUrl(URL,returnres=True,stream=True)

#while 1:

chunk\_size=chunk\_size\*100

for chunk in conn.iter\_content(chunk\_size=chunk\_size):

yield chunk

conn.close()

def download\_file(URL):

return ''.join(download\_chunks(URL))

def validate\_m3u(conn):

''' make sure file is an m3u, and returns the encoding to use. '''

return 'utf8'

mime = conn.headers.get('Content-Type', '').split(';')[0].lower()

if mime == 'application/vnd.apple.mpegurl':

enc = 'utf8'

elif mime == 'audio/mpegurl':

enc = 'iso-8859-1'

elif conn.url.endswith('.m3u8'):

enc = 'utf8'

elif conn.url.endswith('.m3u'):

enc = 'iso-8859-1'

else:

raise Exception("Stream MIME type or file extension not recognized")

if conn.readline().rstrip('\r\n') != '#EXTM3U':

raise Exception("Stream is not in M3U format")

return enc

def gen\_m3u(url, skip\_comments=True):

global cookieJar

conn = getUrl(url,returnres=True )#urllib2.urlopen(url)

redirurl=None

if conn.history:

print 'history'

redirurl=conn.url

enc = validate\_m3u(conn)

#print conn

if redirurl: yield 'f4mredirect:'+redirurl

for line in conn.iter\_lines():#.split('\n'):

line = line.rstrip('\r\n').decode(enc)

if not line:

# blank line

continue

elif line.startswith('#EXT'):

# tag

yield line

elif line.startswith('#'):

# comment

if skip\_comments:

continue

else:

yield line

else:

# media file

yield line

def parse\_m3u\_tag(line):

if ':' not in line:

return line, []

tag, attribstr = line.split(':', 1)

attribs = []

last = 0

quote = False

for i,c in enumerate(attribstr+','):

if c == '"':

quote = not quote

if quote:

continue

if c == ',':

attribs.append(attribstr[last:i])

last = i+1

return tag, attribs

def parse\_kv(attribs, known\_keys=None):

d = {}

for item in attribs:

k, v = item.split('=', 1)

k=k.strip()

v=v.strip().strip('"')

if known\_keys is not None and k not in known\_keys:

raise ValueError("unknown attribute %s"%k)

d[k] = v

return d

def handle\_basic\_m3u(url):

global iv

global key

global USEDec

global gauth

import urlparse,urllib

global callbackDRM

seq = 1

enc = None

nextlen = 5

duration = 5

targetduration=5

aesdone=False

redirurl=url

HOST\_NAME = '127.0.0.1'

PORT\_NUMBER = 55333

vod=False

for line in gen\_m3u(url):

if not line.startswith('#EXT'):

if 1==1:#not line.startswith('http'):

line=urlparse.urljoin(url, line)

newurl='sendvideopart?'+urllib.urlencode({'url': line})

line='http://'+HOST\_NAME+(':%s/'%str(PORT\_NUMBER)) + newurl ##shoud read from config

yield line+'\n'

def player\_pipe(queue, control,file):

while 1:

block = queue.get(block=True)

if block is None: return

file.write(block)

file.flush()

def send\_back(data,file):

file.write(data)

#file.flush()

def downloadInternal(url,file,maxbitrate=0,stopEvent=None , callbackpath="",callbackparam="", testing=False):

global key

global iv

global USEDec

global cookieJar

global clientHeader

global nsplayer

global callbackDRM

if stopEvent and stopEvent.isSet():

return False

dumpfile = None

#dumpfile=open('c:\\temp\\myfile.mp4',"wb")

variants = []

variant = None

veryfirst=True

#url check if requires redirect

redirurl=url

utltext=''

try:

print 'going for url ',url

res=getUrl(url,returnres=True )

print 'here ', res

if res.history:

print 'history is',res.history

redirurl=res.url

url=redirurl

utltext=res.text

res.close()

if testing: return True

except: traceback.print\_exc()

print 'redirurl',redirurl

if 'EXT-X-STREAM-INF' in utltext:

try:

for line in gen\_m3u(redirurl):

if line.startswith('#EXT'):

tag, attribs = parse\_m3u\_tag(line)

if tag == '#EXT-X-STREAM-INF':

variant = attribs

elif variant:

variants.append((line, variant))

variant = None

print 'variants',variants

if len(variants)==0: url=redirurl

if len(variants) == 1:

url = urlparse.urljoin(redirurl, variants[0][0])

elif len(variants) >= 2:

print "More than one variant of the stream was provided."

choice=-1

lastbitrate=0

print 'maxbitrate',maxbitrate

for i, (vurl, vattrs) in enumerate(variants):

print i, vurl,

for attr in vattrs:

key, value = attr.split('=')

key = key.strip()

value = value.strip().strip('"')

if key == 'BANDWIDTH':

print 'bitrate %.2f kbps'%(int(value)/1024.0)

if int(value)<=int(maxbitrate) and int(value)>lastbitrate:

choice=i

lastbitrate=int(value)

elif key == 'PROGRAM-ID':

print 'program %s'%value,

elif key == 'CODECS':

print 'codec %s'%value,

elif key == 'RESOLUTION':

print 'resolution %s'%value,

else:

print "unknown STREAM-INF attribute %s"%key

#raise ValueError("unknown STREAM-INF attribute %s"%key)

print

if choice==-1: choice=0

#choice = int(raw\_input("Selection? "))

print 'choose %d'%choice

url = urlparse.urljoin(redirurl, variants[choice][0])

except:

raise

for chunk in handle\_basic\_m3u(url):

send\_back(chunk,file)