# -\*- coding: utf-8 -\*-

import xml.etree.ElementTree as etree

import base64

from struct import unpack, pack

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 akhds

#import youtube\_dl

#from youtube\_dl.utils import \*

try:

addon\_id = 'plugin.video.f4mTester' # yes its a wrong one but due to settings getting reset

selfAddon = xbmcaddon.Addon(id=addon\_id)

except:

addon\_id = 'script.video.F4mProxy' # yes its a wrong one but due to settings getting reset

selfAddon = xbmcaddon.Addon(id=addon\_id)

\_\_addonname\_\_ = selfAddon.getAddonInfo('name')

\_\_icon\_\_ = selfAddon.getAddonInfo('icon')

downloadPath = xbmc.translatePath(selfAddon.getAddonInfo('profile'))#selfAddon["profile"])

F4Mversion=''

#from Crypto.Cipher import AES

value\_unsafe = '%+&;#'

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

if chr(c) not in value\_unsafe)

def urlencode\_param(value):

"""Minimal URL encoding for query parameter"""

return urllib.quote\_plus(value, safe=VALUE\_SAFE)

class FlvReader(io.BytesIO):

"""

Reader for Flv files

The file format is documented in https://www.adobe.com/devnet/f4v.html

"""

# Utility functions for reading numbers and strings

def read\_unsigned\_long\_long(self):

return unpack('!Q', self.read(8))[0]

def read\_unsigned\_int(self):

return unpack('!I', self.read(4))[0]

def read\_unsigned\_char(self):

return unpack('!B', self.read(1))[0]

def read\_string(self):

res = b''

while True:

char = self.read(1)

if char == b'\x00':

break

res+=char

return res

def read\_box\_info(self):

"""

Read a box and return the info as a tuple: (box\_size, box\_type, box\_data)

"""

real\_size = size = self.read\_unsigned\_int()

box\_type = self.read(4)

header\_end = 8

if size == 1:

real\_size = self.read\_unsigned\_long\_long()

header\_end = 16

return real\_size, box\_type, self.read(real\_size-header\_end)

def read\_asrt(self, debug=False):

version = self.read\_unsigned\_char()

self.read(3) # flags

quality\_entry\_count = self.read\_unsigned\_char()

quality\_modifiers = []

for i in range(quality\_entry\_count):

quality\_modifier = self.read\_string()

quality\_modifiers.append(quality\_modifier)

segment\_run\_count = self.read\_unsigned\_int()

segments = []

#print 'segment\_run\_count',segment\_run\_count

for i in range(segment\_run\_count):

first\_segment = self.read\_unsigned\_int()

fragments\_per\_segment = self.read\_unsigned\_int()

segments.append((first\_segment, fragments\_per\_segment))

#print 'segments',segments

return {'version': version,

'quality\_segment\_modifiers': quality\_modifiers,

'segment\_run': segments,

}

def read\_afrt(self, debug=False):

version = self.read\_unsigned\_char()

self.read(3) # flags

time\_scale = self.read\_unsigned\_int()

quality\_entry\_count = self.read\_unsigned\_char()

quality\_entries = []

for i in range(quality\_entry\_count):

mod = self.read\_string()

quality\_entries.append(mod)

fragments\_count = self.read\_unsigned\_int()

#print 'fragments\_count',fragments\_count

fragments = []

for i in range(fragments\_count):

first = self.read\_unsigned\_int()

first\_ts = self.read\_unsigned\_long\_long()

duration = self.read\_unsigned\_int()

if duration == 0:

discontinuity\_indicator = self.read\_unsigned\_char()

else:

discontinuity\_indicator = None

fragments.append({'first': first,

'ts': first\_ts,

'duration': duration,

'discontinuity\_indicator': discontinuity\_indicator,

})

#print 'fragments',fragments

return {'version': version,

'time\_scale': time\_scale,

'fragments': fragments,

'quality\_entries': quality\_entries,

}

def read\_abst(self, debug=False):

version = self.read\_unsigned\_char()

self.read(3) # flags

bootstrap\_info\_version = self.read\_unsigned\_int()

streamType=self.read\_unsigned\_char()#self.read(1) # Profile,Live,Update,Reserved

islive=False

if (streamType & 0x20) >> 5:

islive=True

print 'LIVE',streamType,islive

time\_scale = self.read\_unsigned\_int()

current\_media\_time = self.read\_unsigned\_long\_long()

smpteTimeCodeOffset = self.read\_unsigned\_long\_long()

movie\_identifier = self.read\_string()

server\_count = self.read\_unsigned\_char()

servers = []

for i in range(server\_count):

server = self.read\_string()

servers.append(server)

quality\_count = self.read\_unsigned\_char()

qualities = []

for i in range(server\_count):

quality = self.read\_string()

qualities.append(server)

drm\_data = self.read\_string()

metadata = self.read\_string()

segments\_count = self.read\_unsigned\_char()

#print 'segments\_count11',segments\_count

segments = []

for i in range(segments\_count):

box\_size, box\_type, box\_data = self.read\_box\_info()

assert box\_type == b'asrt'

segment = FlvReader(box\_data).read\_asrt()

segments.append(segment)

fragments\_run\_count = self.read\_unsigned\_char()

#print 'fragments\_run\_count11',fragments\_run\_count

fragments = []

for i in range(fragments\_run\_count):

# This info is only useful for the player, it doesn't give more info

# for the download process

box\_size, box\_type, box\_data = self.read\_box\_info()

assert box\_type == b'afrt'

fragments.append(FlvReader(box\_data).read\_afrt())

return {'segments': segments,

'movie\_identifier': movie\_identifier,

'drm\_data': drm\_data,

'fragments': fragments,

},islive

def read\_bootstrap\_info(self):

"""

Read the bootstrap information from the stream,

returns a dict with the following keys:

segments: A list of dicts with the following keys

segment\_run: A list of (first\_segment, fragments\_per\_segment) tuples

"""

total\_size, box\_type, box\_data = self.read\_box\_info()

assert box\_type == b'abst'

return FlvReader(box\_data).read\_abst()

def read\_bootstrap\_info(bootstrap\_bytes):

return FlvReader(bootstrap\_bytes).read\_bootstrap\_info()

def build\_fragments\_list(boot\_info, startFromFregment=None, live=True):

""" Return a list of (segment, fragment) for each fragment in the video """

res = []

segment\_run\_table = boot\_info['segments'][0]

print 'segment\_run\_table',segment\_run\_table

# I've only found videos with one segment

#if len(segment\_run\_table['segment\_run'])>1:

# segment\_run\_table['segment\_run']=segment\_run\_table['segment\_run'][-2:] #pick latest

frag\_start = boot\_info['fragments'][0]['fragments']

#print boot\_info['fragments']

# sum(j for i, j in segment\_run\_table['segment\_run'])

first\_frag\_number=frag\_start[0]['first']

last\_frag\_number=frag\_start[-1]['first']

if last\_frag\_number==0:

last\_frag\_number=frag\_start[-2]['first']

endfragment=0

segment\_to\_start=None

for current in range (len(segment\_run\_table['segment\_run'])):

seg,fregCount=segment\_run\_table['segment\_run'][current]

#print 'segmcount',seg,fregCount

if (not live):

frag\_end=last\_frag\_number

else:

frag\_end=first\_frag\_number+fregCount-1

if fregCount>10000:

frag\_end=last\_frag\_number

#if frag\_end

segment\_run\_table['segment\_run'][current]=(seg,fregCount,first\_frag\_number,frag\_end)

if (not startFromFregment==None) and startFromFregment>=first\_frag\_number and startFromFregment<=frag\_end:

segment\_to\_start=current

first\_frag\_number+=fregCount

# print 'current status',segment\_run\_table['segment\_run']

#if we have no index then take the last segment

if segment\_to\_start==None:

segment\_to\_start=len(segment\_run\_table['segment\_run'])-1

#if len(segment\_run\_table['segment\_run'])>2:

# segment\_to\_start=len(segment\_run\_table['segment\_run'])-2;

if live:

startFromFregment=segment\_run\_table['segment\_run'][-1][3]

# if len(boot\_info['fragments'][0]['fragments'])>1: #go bit back

# startFromFregment= boot\_info['fragments'][0]['fragments'][-1]['first']

else:

startFromFregment= boot\_info['fragments'][0]['fragments'][0]['first'] #start from begining

#if len(boot\_info['fragments'][0]['fragments'])>2: #go little bit back

# startFromFregment= boot\_info['fragments'][0]['fragments'][-2]['first']

#print 'startFromFregment',startFromFregment,boot\_info,len(boot\_info['fragments'][0]['fragments'])

#print 'segment\_to\_start',segment\_to\_start

for currentIndex in range (segment\_to\_start,len(segment\_run\_table['segment\_run'])):

currentSegment=segment\_run\_table['segment\_run'][currentIndex]

#print 'currentSegment',currentSegment

(seg,fregCount,frag\_start,frag\_end)=currentSegment

#print 'startFromFregment',startFromFregment,

if (not startFromFregment==None) and startFromFregment>=frag\_start and startFromFregment<=frag\_end:

frag\_start=startFromFregment

#print 'frag\_start',frag\_start,frag\_end

for currentFreg in range(frag\_start,frag\_end+1):

res.append((seg,currentFreg ))

# print 'fragmentlist',res,boot\_info

return res

#totalFrags=sum(j for i, j in segment\_run\_table['segment\_run'])

#lastSegment=segment\_run\_table['segment\_run'][-1]

#lastSegmentStart= lastSegment[0]

#lastSegmentFragCount = lastSegment[1]

#print 'totalFrags',totalFrags

#first\_frag\_number = frag\_start[0]['first']

#startFragOfLastSegment= first\_frag\_number +totalFrags - lastSegmentFragCount

#for (i, frag\_number) in zip(range(1, lastSegmentFragCount+1), itertools.count(startFragOfLastSegment)):

# res.append((lastSegmentStart,frag\_number )) #this was i, i am using first segement start

#return res

#segment\_run\_entry = segment\_run\_table['segment\_run'][0]

#print 'segment\_run\_entry',segment\_run\_entry,segment\_run\_table

#n\_frags = segment\_run\_entry[1]

#startingPoint = segment\_run\_entry[0]

#fragment\_run\_entry\_table = boot\_info['fragments'][0]['fragments']

#frag\_entry\_index = 0

#first\_frag\_number = fragment\_run\_entry\_table[0]['first']

#first\_frag\_number=(startingPoint\*n\_frags) -(n\_frags)+1

#print 'THENUMBERS',startingPoint,n\_frags,first\_frag\_number

#for (i, frag\_number) in zip(range(1, n\_frags+1), itertools.count(first\_frag\_number)):

# res.append((startingPoint,frag\_number )) #this was i, i am using first segement start

#return res

def join(base,url):

join = urlparse.urljoin(base,url)

url = urlparse.urlparse(join)

path = posixpath.normpath(url[2])

return urlparse.urlunparse(

(url.scheme,url.netloc,path,url.params,url.query,url.fragment)

)

def \_add\_ns(prop):

#print 'F4Mversion',F4Mversion

return '{http://ns.adobe.com/f4m/%s}%s' %(F4Mversion, prop)

#class ReallyQuietDownloader(youtube\_dl.FileDownloader):

# def to\_screen(sef, \*args, \*\*kargs):

# pass

class F4MDownloader():

"""

A downloader for f4m manifests or AdobeHDS.

"""

outputfile =''

clientHeader=None

cookieJar=cookielib.LWPCookieJar()

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

self.init\_done=False

def getUrl(self,url, ischunkDownloading=False):

try:

post=None

print 'url',url

#openner = urllib2.build\_opener(urllib2.HTTPHandler, urllib2.HTTPSHandler)

cookie\_handler = urllib2.HTTPCookieProcessor(self.cookieJar)

openner = urllib2.build\_opener(cookie\_handler, urllib2.HTTPBasicAuthHandler(), urllib2.HTTPHandler())

if post:

req = urllib2.Request(url, post)

else:

req = urllib2.Request(url)

ua\_header=False

if self.clientHeader:

for n,v in self.clientHeader:

req.add\_header(n,v)

if n=='User-Agent':

ua\_header=True

if not ua\_header:

req.add\_header('User-Agent','Mozilla/5.0 (Windows NT 6.1; Win64; x64; Trident/7.0; rv:11.0) like Gecko')

#response = urllib2.urlopen(req)

if self.proxy and ( (not ischunkDownloading) or self.use\_proxy\_for\_chunks ):

req.set\_proxy(self.proxy, 'http')

response = openner.open(req)

data=response.read()

return data

except:

print 'Error in getUrl'

traceback.print\_exc()

return None

def \_write\_flv\_header2(self, stream):

"""Writes the FLV header and the metadata to stream"""

# FLV header

stream.write(b'FLV\x01')

stream.write(b'\x01')

stream.write(b'\x00\x00\x00\x09')

# FLV File body

stream.write(b'\x00\x00\x00\x09')

def \_write\_flv\_header(self, stream, metadata):

"""Writes the FLV header and the metadata to stream"""

# FLV header

stream.write(b'FLV\x01')

stream.write(b'\x05')

stream.write(b'\x00\x00\x00\x09')

# FLV File body

stream.write(b'\x00\x00\x00\x00')

# FLVTAG

if metadata:

stream.write(b'\x12') # Script data

stream.write(pack('!L',len(metadata))[1:]) # Size of the metadata with 3 bytes

stream.write(b'\x00\x00\x00\x00\x00\x00\x00')

stream.write(metadata)

# All this magic numbers have been extracted from the output file

# produced by AdobeHDS.php (https://github.com/K-S-V/Scripts)

stream.write(b'\x00\x00\x01\x73')

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

try:

self.init\_done=False

self.total\_frags=0

self.init\_url=url

self.clientHeader=None

self.status='init'

self.proxy = proxy

self.auth=auth

#self.auth="pvtoken=exp%3D9999999999%7Eacl%3D%252f%252a%7Edata%3DZXhwPTE0MDYzMDMxMTV+YWNsPSUyZip+ZGF0YT1wdmMsc35obWFjPWQxODA5MWVkYTQ4NDI3NjFjODhjOWQwY2QxNTk3YTI0MWQwOWYwNWI1N2ZmMDE0ZjcxN2QyMTVjZTJkNmJjMDQ%3D%2196e4sdLWrezE46RaCBzzP43/LEM5en2KujAosbeDimQ%3D%7Ehmac%3DACF8A1E4467676C9BCE2721CA5EFF840BD6ED1780046954039373A3B0D942ADC&hdntl=exp=1406303115~acl=%2f\*~data=hdntl~hmac=4ab96fa533fd7c40204e487bfc7befaf31dd1f49c27eb1f610673fed9ff97a5f&als=0,2,0,0,0,NaN,0,0,0,37,f,52293145.57,52293155.9,t,s,GARWLHLMHNGA,2.11.3,37&hdcore=2.11.3"

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

self.auth=''

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

self.proxy=None

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

self.out\_stream=out\_stream

self.g\_stopEvent=g\_stopEvent

self.maxbitrate=maxbitrate

if '|' in url:

sp = url.split('|')

url = sp[0]

self.clientHeader = sp[1]

self.clientHeader= urlparse.parse\_qsl(self.clientHeader)

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

self.status='init done'

self.url=url

self.swf=swf

#self.downloadInternal( url)

return self.preDownoload()

#os.remove(self.outputfile)

except:

traceback.print\_exc()

self.status='finished'

return False

def preDownoload(self):

global F4Mversion

try:

self.seqNumber=0

self.live=False #todo find if its Live or not

man\_url = self.url

url=self.url

print 'Downloading f4m manifest'

manifest = self.getUrl(man\_url)#.read()

if not manifest:

return False

print len(manifest)

try:

print manifest

except: pass

self.status='manifest done'

#self.report\_destination(filename)

#dl = ReallyQuietDownloader(self.ydl, {'continuedl': True, 'quiet': True, 'noprogress':True})

version\_fine="xmlns=\".\*?\/([0-9].\*?)\""

F4Mversion =re.findall(version\_fine, manifest)[0]

#print F4Mversion,\_add\_ns('media')

auth\_patt='<pv-2.0>(.\*?)<'

auth\_obj =re.findall(auth\_patt, manifest)

self.auth20=''

if auth\_obj and len(auth\_obj)>0:

self.auth20=auth\_obj[0] #not doing anything for time being

print 'auth got from xml',self.auth,self.auth20

#quick for one example where the xml was wrong.

if '\"bootstrapInfoId' in manifest:

manifest=manifest.replace('\"bootstrapInfoId','\" bootstrapInfoId')

doc = etree.fromstring(manifest)

print doc

# Added the-one 05082014

# START

# Check if manifest defines a baseURL tag

baseURL\_tag = doc.find(\_add\_ns('baseURL'))

if baseURL\_tag != None:

man\_url = baseURL\_tag.text

url = man\_url

self.url = url

print 'base url defined as: %s' % man\_url

# END

try:

#formats = [(int(f.attrib.get('bitrate', -1)),f) for f in doc.findall(\_add\_ns('media'))]

formats=[]

for f in doc.findall(\_add\_ns('media')):

vtype=f.attrib.get('type', '')

if f.attrib.get('type', '')=='video' or vtype=='' :

formats.append([int(f.attrib.get('bitrate', -1)),f])

print 'format works',formats

except:

formats=[(int(0),f) for f in doc.findall(\_add\_ns('media'))]

#print 'formats',formats

formats = sorted(formats, key=lambda f: f[0])

if self.maxbitrate==0:

rate, media = formats[-1]

elif self.maxbitrate==-1:

rate, media = formats[0]

else: #find bitrate

brselected=None

rate, media=None,None

for r, m in formats:

if r<=self.maxbitrate:

rate, media=r,m

else:

break

if media==None:

rate, media = formats[-1]

dest\_stream = self.out\_stream

print 'rate selected',rate

self.metadata=None

try:

self.metadata = base64.b64decode(media.find(\_add\_ns('metadata')).text)

print 'metadata stream read done'#,media.find(\_add\_ns('metadata')).text

#self.\_write\_flv\_header(dest\_stream, metadata)

#dest\_stream.flush()

except: pass

# Modified the-one 05082014

# START

# url and href can be used interchangeably

# so if url attribute is not present

# check for href attribute

try:

mediaUrl=media.attrib['url']

except:

mediaUrl=media.attrib['href']

# END

# Added the-one 05082014

# START

# if media url/href points to another f4m file

if '.f4m' in mediaUrl:

sub\_f4m\_url = join(man\_url,mediaUrl)

print 'media points to another f4m file: %s' % sub\_f4m\_url

print 'Downloading f4m sub manifest'

sub\_manifest = self.getUrl(sub\_f4m\_url)#.read()

if not sub\_manifest:

return False

print len(sub\_manifest)

try:

print sub\_manifest

except: pass

self.status='sub manifest done'

F4Mversion =re.findall(version\_fine, sub\_manifest)[0]

doc = etree.fromstring(sub\_manifest)

print doc

media = doc.find(\_add\_ns('media'))

if media == None:

return False

try:

self.metadata = base64.b64decode(media.find(\_add\_ns('metadata')).text)

print 'metadata stream read done'

except: pass

try:

mediaUrl=media.attrib['url']

except:

mediaUrl=media.attrib['href']

# END

try:

bootStrapID = media.attrib['bootstrapInfoId']

except: bootStrapID='xx'

#print 'mediaUrl',mediaUrl

base\_url = join(man\_url,mediaUrl)#compat\_urlparse.urljoin(man\_url,media.attrib['url'])

keybase\_url=join(man\_url,'key\_')

if mediaUrl.endswith('/') and not base\_url.endswith('/'):

base\_url += '/'

self.base\_url=base\_url

self.keybase\_url=keybase\_url

bsArray=doc.findall(\_add\_ns('bootstrapInfo'))

print 'bootStrapID',bootStrapID

#bootStrapID='bootstrap\_450'

bootstrap=self.getBootStrapWithId(bsArray,bootStrapID)

if bootstrap==None: #if not available then find any!

print 'bootStrapID NOT Found'

bootstrap=doc.findall(\_add\_ns('bootstrapInfo'))[0]

else:

print 'found bootstrap with id',bootstrap

#print 'bootstrap',bootstrap

bootstrapURL1=''

try:

bootstrapURL1=bootstrap.attrib['url']

except: pass

bootstrapURL=''

bootstrapData=None

queryString=None

if bootstrapURL1=='':

bootstrapData=base64.b64decode(doc.findall(\_add\_ns('bootstrapInfo'))[0].text)

#

else:

from urlparse import urlparse

queryString = urlparse(url).query

print 'queryString11',queryString

if len(queryString)==0: queryString=None

if queryString==None or '?' in bootstrap.attrib['url']:

bootstrapURL = join(man\_url,bootstrap.attrib['url'])# take out querystring for later

queryString = urlparse(bootstrapURL).query

print 'queryString override',queryString

if len(queryString)==0:

queryString=None

if len(self.auth)>0:

bootstrapURL+='?'+self.auth

queryString=self.auth#self.\_pv\_params('',self.auth20)#not in use

elif len(self.auth20)>0:

queryString=self.\_pv\_params(self.swf,self.auth20)

bootstrapURL+='?'+queryString

else:

print 'queryString!!',queryString

bootstrapURL = join(man\_url,bootstrap.attrib['url'])+'?'+queryString

if len(self.auth)>0:

authval=self.auth#self.\_pv\_params('',self.auth20)#not in use

bootstrapURL = join(man\_url,bootstrap.attrib['url'])+'?'+authval

queryString=authval

elif len(self.auth20)>0:

authval=self.\_pv\_params(self.swf,self.auth20)#requires swf param

bootstrapURL = join(man\_url,bootstrap.attrib['url'])+'?'+authval

queryString=authval

print 'bootstrapURL',bootstrapURL

if queryString==None:

queryString=''

self.bootstrapURL=bootstrapURL

self.queryString=queryString

self.bootstrap, self.boot\_info, self.fragments\_list,self.total\_frags=self.readBootStrapInfo(bootstrapURL,bootstrapData)

self.init\_done=True

return True

except:

traceback.print\_exc()

return False

def readAKKey(self, data):

messageKeyExists=False

key=""

firstByte=ord(data[0])

pos=1

returnIV=None

if firstByte==12: #version12

pos+=4+4+2+1;

# print 'indeedddd',firstByte

# print 'data',repr(data)

messageByte=ord(data[pos])

pos+=1

messageKeyExists=(messageByte & 4) > 0;

messageIV=(messageByte & 2) > 0;

if messageIV:

pos+=16

# print 'IV exists'

if messageKeyExists:

# print 'Message Key exists!!!'

returnIV=data[pos-16:pos]

d = str(data[pos]);

pos+=1

key = d;

while(d != '\x00'):

d = str(data[pos]);

pos+=1

if d != '\x00':

key+= d;

else:

print 'SOMETHING WRONG.... got other than 12'

print 1/0#shouldn't come where

return messageKeyExists, key,pos,returnIV

def getFrames(self,box\_data, remainingdata):

frames=[]

KeepProcessing = False;

currentStep= 0;

tagLen = 0;

if(box\_data):

if remainingdata and len(remainingdata)>0:

box\_data=remainingdata+box\_data

remainingdata=None

lookForTagStart = 0;

KeepProcessing = True;

while(KeepProcessing and lookForTagStart<len(box\_data)):

currentStep = ord(box\_data[lookForTagStart]);

tagLen = ord(box\_data[lookForTagStart + 1]) << 16 | ord(box\_data[lookForTagStart + 2]) << 8 | ord(box\_data[lookForTagStart + 3]) & 255;

nextTag = lookForTagStart + 11 + tagLen + 4

if (nextTag > len(box\_data) and currentStep > 0):

remainingdata = [];

remainingdata=box\_data[lookForTagStart:]

KeepProcessing = False;

elif (currentStep > 0):

chunk = []

chunk=box\_data[lookForTagStart:lookForTagStart+tagLen + 11 + 4]

frames.append((1,chunk))

elif (currentStep == 0):

KeepProcessing = False;

#if nextTag==len(box\_data):

# KeepProcessing=False

#print nextTag, len(box\_data)

lookForTagStart = nextTag;

return frames,remainingdata

# #def AES(self,key):

# return Rijndael(key, keySize=16, blockSize=16, padding=padWithPadLen())

# def AES\_CBC(self,key):

# return CBC(blockCipherInstance=AES(key))

def addBytesToOutput(self,prefix,data,post,segmentid,buffer):

dataLen=0

if data and len(data)>0:

dataLen=len(data)

#print 'Incomming',repr(prefix)

prefix=list(prefix)

prefix[3]=chr(dataLen & 255)

prefix[2]=chr(dataLen >> 8 & 255);

prefix[1]=chr(dataLen >> 16 & 255);

#print repr(prefix)

prefix=''.join(prefix)

#print repr(prefix)

#print len(prefix)

finalArray=prefix

if data and len(data)>0:

finalArray+=data

if post and len(post):

finalArray+=post

# with open("c:\\temp\\myfile.mp4", 'a+b') as output:

# output.write(finalArray)

lenReturned=len(finalArray)

buffer.write(finalArray)

buffer.flush()

return lenReturned

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

try:

self.status='download Starting'

self.downloadInternal(self.url,dest\_stream,segmentToStart,totalSegmentToSend)

except:

traceback.print\_exc()

try:

akhds.cleanup()

except:pass

self.status='finished'

def downloadInternal(self,url,dest\_stream ,segmentToStart=None,totalSegmentToSend=0):

global F4Mversion

try:

#dest\_stream = self.out\_stream

queryString=self.queryString

print 'segmentToStart',segmentToStart

if self.live or segmentToStart==0 or segmentToStart==None:

print 'writing metadata'#,len(self.metadata)

self.\_write\_flv\_header(dest\_stream, self.metadata)

dest\_stream.flush()

#elif segmentToStart>0 and not self.live:

# self.\_write\_flv\_header2(dest\_stream)

# dest\_stream.flush()

url=self.url

bootstrap, boot\_info, fragments\_list,total\_frags=(self.bootstrap, self.boot\_info, self.fragments\_list,self.total\_frags)

print boot\_info, fragments\_list,total\_frags

self.status='bootstrap done'

self.status='file created'

self.downloaded\_bytes = 0

self.bytes\_in\_disk = 0

self.frag\_counter = 0

start = time.time()

frags\_filenames = []

self.seqNumber=0

if segmentToStart and not self.live :

self.seqNumber=segmentToStart

if self.seqNumber>=total\_frags:

self.seqNumber=total\_frags-1

#for (seg\_i, frag\_i) in fragments\_list:

#for seqNumber in range(0,len(fragments\_list)):

self.segmentAvailable=0

frameSent=0

keyValue=""

keyData=None

firstPacket=True

remainingFrameData=None

decrypter=None

errors=0

file=0

lastIV=None

AKSession=None

while True:

#if not self.live:

# \_write\_flv\_header2

try:

if self.g\_stopEvent.isSet():

return

except: pass

seg\_i, frag\_i=fragments\_list[self.seqNumber]

self.seqNumber+=1

frameSent+=1

name = u'Seg%d-Frag%d' % (seg\_i, frag\_i)

#print 'base\_url',base\_url,name

if AKSession:

name+=AKSession

url = self.base\_url + name

if queryString and '?' not in url:

url+='?'+queryString

elif '?' in self.base\_url:

url = self.base\_url.split('?')[0] + name+'?'+self.base\_url.split('?')[1]

#print(url),base\_url,name

#frag\_filename = u'%s-%s' % (tmpfilename, name)

#success = dl.\_do\_download(frag\_filename, {'url': url})

print 'downloading....',url

success=False

urlTry=0

while not success and urlTry<5:

success = self.getUrl(url,True)

if not success: xbmc.sleep(300)

urlTry+=1

print 'downloaded',not success==None,url

if not success:

return False

#with open(frag\_filename, 'rb') as down:

if 1==1:

down\_data = success#down.read()

reader = FlvReader(down\_data)

while True:

\_, box\_type, box\_data = reader.read\_box\_info()

print 'box\_type',box\_type,len(box\_data)

#if box\_type == b'afra':

# dest\_stream.write(box\_data)

# dest\_stream.flush()

# break

if box\_type == b'mdat':

isDrm=True if ord(box\_data[0])&1 else False

boxlength=len(box\_data)

seglen=0

file+=1

# if file>6: print 1/0

skip=False

doDecrypt=False

# print 'first byte',repr(box\_data[0]),'kk'

isAkamaiEncrypted=True if ord(box\_data[0])==11 or ord(box\_data[0])==10 else False

if isAkamaiEncrypted:

# print 'Total MDAT count',len(box\_data), len(box\_data)%16

\_loc8\_ = ord(box\_data[1]) << 16 | ord(box\_data[2]) << 8 | ord(box\_data[3]) & 255;

\_loc9\_ = box\_data[11:11+\_loc8\_]

# print 'this is encrypted',len(\_loc9\_),\_loc8\_,repr(box\_data[1:70])

keyExists,Key,dataread,lastIV=self.readAKKey(\_loc9\_)

if keyExists:

# print 'key exists and its len is ',\_loc8\_,repr(Key)

doDecrypt=True

keyValueNew=Key.split('key\_')[1]

# print 'previous key is'+keyValue,'new key is',keyValueNew

if keyValue=="":

keyValue="\_"+keyValueNew

AKSession=keyValue

keyurl = self.keybase\_url +keyValueNew

if queryString and '?' not in keyurl:

keyurl+='?'+queryString+'&guid=CHRLRCMRHGUD'

print 'the key url is ',keyurl,'thanks'

keyData=self.getUrl(keyurl,False)

skip=False

firstPacket=True

elif not keyValue=="\_"+keyValueNew:

keyValue="\_"+keyValueNew#take new key

AKSession=keyValue

keyurl = self.keybase\_url +keyValueNew

if queryString and '?' not in keyurl:

keyurl+='?'+queryString+'&guid=CHRLRCMRHGUD'

keyData=self.getUrl(keyurl,False)

firstPacket=True

#todo decryptit! and put it in box\_data

#print 'before skip'

if skip:

break;

if keyData:

# print 'key data is',repr(keyData),len(keyData)

#do decrypt here. frame by frame

#now generate frames

#put remaining in remaining

#for each frame decrypt and write and flush

try:

frames=[]

# print 'before frames data', repr(box\_data[0:70])

frames,remainingFrameData=self.getFrames(box\_data,remainingFrameData)

# print 'after frames data first frame', repr(frames[0][0:70])

#print 'frames',frames

cleanup=False

for frame in frames:

data=frame[1]

datalen=ord(data[1]) << 16 | ord(data[2]) << 8 | ord(data[3]) & 255;

preFrame=len(data)

#print 'samp>',len(data),datalen,ord(data[0]) ,'<samp'

if firstPacket:

firstPacket=False

# data=data[0:datalen]

#print 'first>',len(data),ord(data[0]),datalen,'<first'

# else:

if 1==1:

#if not not key frame then decrypt else

firstByte=ord(data[0])

frameHeader=data[:11]

framePad=data[11 + datalen:11 + datalen+4];

if firstByte==10 or firstByte==11:

if firstByte==10:

frameHeader = list(frameHeader)

frameHeader[0]=chr(8)

frameHeader=''.join(frameHeader)

if firstByte==11:

frameHeader = list(frameHeader)

frameHeader[0]=chr(9)

frameHeader=''.join(frameHeader)

data=data[11:11+datalen]

#print 'sub>',len(data),firstByte,datalen,datalen%16,len(data)%16 ,'<sub'

keyExistsNew,KeyNew,dataread,ignoreIV=self.readAKKey(data)

# print 'dataread',dataread,keyExistsNew,KeyNew,ignoreIV

try:

akhds.init()

data=akhds.tagDecrypt(data,keyData)

except:

print 'decryption error'

errors+=1

traceback.print\_exc()

if errors>10: print 1/0

# print 'pre return size %d, %d %d'%(len(frameHeader),len(data), len(framePad))

seglen1=self.addBytesToOutput(frameHeader,data,framePad,1,dest\_stream)

seglen+=seglen1

# print 'pre frame %d, after %d'%(preFrame,seglen1)

else:

print 'hmm no 10 or 11?'

# print 'pre return size %d, %d %d'%(len(frameHeader),len(data), len(framePad))

seglen1=self.addBytesToOutput(frameHeader,None,None,1,dest\_stream)

seglen+=seglen1

# print 'pre frame %d, after %d'%(preFrame,seglen1)

#est\_stream.write(data)

#dest\_stream.flush()

#dest\_stream.write(self.decryptData(data,keyData))

#dest\_stream.flush()

except:

print traceback.print\_exc()

self.g\_stopEvent.set()

else:

dest\_stream.write(box\_data)

dest\_stream.flush()

print 'box length is %d and seg total is %d'%(boxlength,seglen)

break

# Using the following code may fix some videos, but

# only in mplayer, VLC won't play the sound.

# mdat\_reader = FlvReader(box\_data)

# media\_type = mdat\_reader.read\_unsigned\_char()

# while True:

# if mdat\_reader.read\_unsigned\_char() == media\_type:

# if mdat\_reader.read\_unsigned\_char() == 0x00:

# break

# dest\_stream.write(pack('!B', media\_type))

# dest\_stream.write(b'\x00')

# dest\_stream.write(mdat\_reader.read())

# break

self.status='play'

if self.seqNumber==len(fragments\_list) or (totalSegmentToSend>0 and frameSent==totalSegmentToSend):

if not self.live:

break

self.seqNumber=0

#todo if the url not available then get manifest and get the data again

total\_frags=None

try:

bootstrap, boot\_info, fragments\_list,total\_frags=self.readBootStrapInfo(self.bootstrapURL,None,updateMode=True,lastSegment=seg\_i, lastFragement=frag\_i)

except:

traceback.print\_exc()

pass

if total\_frags==None:

break

del self.downloaded\_bytes

del self.frag\_counter

except:

traceback.print\_exc()

return

def getBootStrapWithId (self,BSarray, id):

try:

for bs in BSarray:

print 'compare val is ',bs.attrib['id'], 'id', id

if bs.attrib['id']==id:

print 'gotcha'

return bs

except: pass

return None

def readBootStrapInfo(self,bootstrapUrl,bootStrapData, updateMode=False, lastFragement=None,lastSegment=None):

try:

retries=0

while retries<=10:

try:

if self.g\_stopEvent.isSet():

print 'event is set. returning'

return

except: pass

if bootStrapData==None:

bootStrapData =self.getUrl(bootstrapUrl)

if bootStrapData==None:

retries+=1

continue

#print 'bootstrapData',len(bootStrapData)

bootstrap = bootStrapData#base64.b64decode(bootStrapData)#doc.findall(\_add\_ns('bootstrapInfo'))[0].text)

#print 'boot stream read done'

boot\_info,self.live = read\_bootstrap\_info(bootstrap)

#print 'boot\_info read done',boot\_info

newFragement=None

if not lastFragement==None:

newFragement=lastFragement+1

fragments\_list = build\_fragments\_list(boot\_info,newFragement,self.live)

total\_frags = len(fragments\_list)

#print 'fragments\_list',fragments\_list, newFragement

#print lastSegment

if updateMode and (len(fragments\_list)==0 or ( newFragement and newFragement>fragments\_list[0][1])):

#todo check lastFragement to see if we got valid data

print 'retrying......'

bootStrapData=None

retries+=1

xbmc.sleep(2000)

continue

return bootstrap, boot\_info, fragments\_list,total\_frags

except:

traceback.print\_exc()

def \_pv\_params(self, pvswf, pv):

"""Returns any parameters needed for Akamai HD player verification.

Algorithm originally documented by KSV, source:

http://stream-recorder.com/forum/showpost.php?p=43761&postcount=13

"""

#return pv;

#pv="ZXhwPTE0NDAxNTUyODJ+YWNsPSUyZip+ZGF0YT1wdmMsc35obWFjPWMyZjk4MmVjZjFjODQyM2IzZDkxMzExMjNmY2ExN2U4Y2UwMjU4NWFhODg3MWFjYzM5YmI0MmVlNTYxYzM5ODc="

# pv="ZXhwPTE0NDAzMjc3ODF+YWNsPSUyZip+ZGF0YT1wdmMsc35obWFjPTYyYTE2MzU2MTNjZTI4ZWI2MTg0MmRjYjFlZTZlYTYwYTA5NWUzZDczNTQ5MTQ1ZDVkNTc0M2M2Njk5MDJjNjY="

# pv="ZXhwPTE0Mzk2MDgzMTl+YWNsPSUyZip+ZGF0YT1wdmMsc35obWFjPTExYTJiNzQ4NjQyYmY1M2VlNzk5MzhhNTMzNjc1MTAzZjk2NWViOGVhODY4MzUwODkwZGM1MjVmNjI3ODM4MzQ="

try:

data, hdntl = pv.split(";")

except ValueError:

data = pv

hdntl = ""

print 'DATA IS',data

print 'hdntl IS',hdntl

if data=="": return hdntl

first\_stage\_msg=binascii.unhexlify('056377146640142763057567157640125041016376130175171220177717044510157134116364123221072012122137150351003442036164015632157517073355151142067436113220106435137171174171127530157325044270025004')

first\_stage\_key=data

hash\_data=""

if pvswf is None:

print 'swf required for pv2 decryption'

pvswf=""

if pvswf.startswith('http'):

import hashlib

h=hashlib.md5()

h.update(pvswf)

hashkey=""+str(h.hexdigest())

existinghash=str(selfAddon.getSetting(hashkey))

#print 'existinghash',hashkey

#print 'existinghashval',existinghash

if len(existinghash)==0:

swf = self.getUrl(pvswf,False)

hash = hashlib.sha256()

hash.update(self.swfdecompress(swf))

hash = base64.b64encode(hash.digest()).decode("ascii")

#print hashkey,hash

selfAddon.setSetting(hashkey, str(hash))

#print 'getting back',str(selfAddon.getSetting(hashkey))

else:

hash=existinghash

else:

hash=pvswf # the incoming is the hash!

print 'hash',hash

# shouldhash="AFe6zmDCNudrcFNyePaAzAn/KRT5ES99ql4SNqldM2I="

# if shouldhash==hash:

# print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*HASH MATCH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

# else:

# print '\*\*\*\*\*\*\*\*\* NOOOOOOOOOOOOOOOOOOOOTTTTTTTTTTTTTTTTT\*\*\*\*\*\*\*\*\*\*'

second\_stage\_key = hmac.new(first\_stage\_key,first\_stage\_msg , sha256).digest()

# second\_stage\_data=hash\_data #

second\_stage\_data=base64.b64decode( hash)

buffer="106,45,165,20,106,45,165,20,38,45,165,87,11,98,228,14,107,89,233,25,101,36,223,76,97,28,175,18,23,86,164,6,1,56,157,64,123,58,186,100,54,34,184,14,3,44,164,20,106,6,222,84,122,45,165,20,106,28,196,84,122,111,183,84,122,45,165,20,106,45,165,20,106,45,165,20,106,45,165,20,106,45,165,20,106,45,165,20,106,45,165,20,106,45,165,20"

buffer=buffer.split(',');

second\_stage\_data+=chr(int(buffer[len(second\_stage\_data)]))

# print len(second\_stage\_data),repr(second\_stage\_data)

third\_stage\_key= hmac.new(second\_stage\_key, second\_stage\_data, sha256).digest()

#hash=shouldhash

msg = "exp=9999999999~acl=%2f%2a~data={0}!{1}".format(data, hash)

auth = hmac.new(third\_stage\_key, msg.encode("ascii"), sha256)

pvtoken = "{0}~hmac={1}".format(msg, auth.hexdigest())

# The "hdntl" parameter can be accepted as a cookie or passed in the

# query string, but the "pvtoken" parameter can only be in the query

# string

print 'pvtoken',pvtoken

params=urllib.urlencode({'pvtoken':pvtoken})+'&'+hdntl+'&hdcore=2.11.3'

params=params.replace('%2B','+')

params=params.replace('%2F','/')

print params

return params

def swfdecompress(self,data):

if data[:3] == b"CWS":

data = b"F" + data[1:8] + zlib.decompress(data[8:])

return data