################################################################################

# Copyright (C) 2013 Sean Poyser #

# #

# This Program is free software; you can redistribute it and/or modify #

# it under the terms of the GNU General Public License as published by #

# the Free Software Foundation; either version 2, or (at your option) #

# any later version. #

# #

# This Program is distributed in the hope that it will be useful, #

# but WITHOUT ANY WARRANTY; without even the implied warranty of #

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the #

# GNU General Public License for more details. #

# #

# You should have received a copy of the GNU General Public License #

# along with XBMC; see the file COPYING. If not, write to #

# the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA. #

# http://www.gnu.org/copyleft/gpl.html #

#################################################################################

# 5: "240p h263 flv container",

# 18: "360p h264 mp4 container | 270 for rtmpe?",

# 22: "720p h264 mp4 container",

# 26: "???",

# 33: "???",

# 34: "360p h264 flv container",

# 35: "480p h264 flv container",

# 37: "1080p h264 mp4 container",

# 38: "720p vp8 webm container",

# 43: "360p h264 flv container",

# 44: "480p vp8 webm container",

# 45: "720p vp8 webm container",

# 46: "520p vp8 webm stereo",

# 59: "480 for rtmpe",

# 78: "seems to be around 400 for rtmpe",

# 82: "360p h264 stereo",

# 83: "240p h264 stereo",

# 84: "720p h264 stereo",

# 85: "520p h264 stereo",

# 100: "360p vp8 webm stereo",

# 101: "480p vp8 webm stereo",

# 102: "720p vp8 webm stereo",

# 120: "hd720",

# 121: "hd1080"

import re

import urllib2

import urllib

import cgi

import HTMLParser

import xbmcgui

try: import simplejson as json

except ImportError: import json

dp = xbmcgui.DialogProgress()

MAX\_REC\_DEPTH = 5

def Clean(text):

text = text.replace('&#8211;', '-')

text = text.replace('&#8217;', '\'')

text = text.replace('&#8220;', '"')

text = text.replace('&#8221;', '"')

text = text.replace('&#39;', '\'')

text = text.replace('<b>', '')

text = text.replace('</b>', '')

text = text.replace('&amp;', '&')

text = text.replace('\ufeff', '')

return text

def PlayVideo(id, forcePlayer=False):

import sys

dp.create("Loading video",'','Please Wait','')

video, links = GetVideoInformation(id)

if 'best' not in video:

return False

url = video['best']

title = video['title']

image = video['thumbnail']

liz = xbmcgui.ListItem(title, iconImage=image, thumbnailImage=image)

liz.setInfo( type="Video", infoLabels={ "Title": title} )

if forcePlayer or len(sys.argv) < 2 or int(sys.argv[1]) == -1:

import xbmc

pl = xbmc.PlayList(xbmc.PLAYLIST\_VIDEO)

pl.clear()

pl.add(url, liz)

dp.close()

xbmc.Player().play(pl, windowed=False)

else:

import xbmcplugin

liz.setPath(url)

xbmcplugin.setResolvedUrl(int(sys.argv[1]), True, liz)

return True

def PlayVideoB(id, forcePlayer=False):

import sys

dp.create("Loading video",'<>','Please Wait','<>')

video, links = GetVideoInformation(id)

if 'best' not in video:

return False

url = video['best']

title = video['title']

image = video['thumbnail']

liz = xbmcgui.ListItem(title, iconImage=image, thumbnailImage=image)

liz.setInfo( type="Video", infoLabels={ "Title": title} )

import xbmc

pl = xbmc.PlayList(xbmc.PLAYLIST\_VIDEO)

pl.clear()

pl.add(url, liz)

dp.close()

xbmc.Player().play(pl, windowed=False)

def GetVideoInformation(id):

#id = 'H7iQ4sAf0OE' #test for HLSVP

#id = 'ofHlUJuw8Ak' #test for stereo

#id = 'ifZkeuSrNRc' #account closed

#id = 'M7FIvfx5J10'

#id = 'n-D1EB74Ckg' #vevo

#id = 'lVMWEheQ2hU' #vevo

video = {}

links = []

try: video, links = GetVideoInfo(id)

except : pass

return video, links

def GetVideoInfo(id):

url = 'http://www.youtube.com/watch?v=%s&safeSearch=none' % id

html = FetchPage(url)

video, links = Scrape(html)

video['videoid'] = id

video['thumbnail'] = "http://i.ytimg.com/vi/%s/0.jpg" % video['videoid']

video['title'] = GetVideoTitle(html)

if len(links) == 0:

if 'hlsvp' in video:

video['best'] = video['hlsvp']

else:

video['best'] = links[0][1]

return video, links

def GetVideoTitle(html):

try: return Clean(re.compile('<meta name="title" content="(.+?)">').search(html).groups(1)[0])

except: pass

return 'YouTube Video'

def Scrape(html):

stereo = [82, 83, 84, 85, 100, 101, 102]

video = {}

links = []

flashvars = ExtractFlashVars(html)

if not flashvars.has\_key(u"url\_encoded\_fmt\_stream\_map"):

return video, links

if flashvars.has\_key(u"ttsurl"):

video[u"ttsurl"] = flashvars[u"ttsurl"]

if flashvars.has\_key(u"hlsvp"):

video[u"hlsvp"] = flashvars[u"hlsvp"]

for url\_desc in flashvars[u"url\_encoded\_fmt\_stream\_map"].split(u","):

url\_desc\_map = cgi.parse\_qs(url\_desc)

if not (url\_desc\_map.has\_key(u"url") or url\_desc\_map.has\_key(u"stream")):

continue

key = int(url\_desc\_map[u"itag"][0])

url = u""

if url\_desc\_map.has\_key(u"url"):

url = urllib.unquote(url\_desc\_map[u"url"][0])

elif url\_desc\_map.has\_key(u"conn") and url\_desc\_map.has\_key(u"stream"):

url = urllib.unquote(url\_desc\_map[u"conn"][0])

if url.rfind("/") < len(url) -1:

url = url + "/"

url = url + urllib.unquote(url\_desc\_map[u"stream"][0])

elif url\_desc\_map.has\_key(u"stream") and not url\_desc\_map.has\_key(u"conn"):

url = urllib.unquote(url\_desc\_map[u"stream"][0])

if url\_desc\_map.has\_key(u"sig"):

url = url + u"&signature=" + url\_desc\_map[u"sig"][0]

elif url\_desc\_map.has\_key(u"s"):

sig = url\_desc\_map[u"s"][0]

#url = url + u"&signature=" + DecryptSignature(sig)

flashvars = ExtractFlashVars(html, assets=True)

js = flashvars[u"js"]

url += u"&signature=" + DecryptSignatureNew(sig, js)

if key not in stereo:

links.append([key, url])

#links.sort(reverse=True)

return video, links

def DecryptSignature(s):

''' use decryption solution by Youtube-DL project '''

if len(s) == 88:

return s[48] + s[81:67:-1] + s[82] + s[66:62:-1] + s[85] + s[61:48:-1] + s[67] + s[47:12:-1] + s[3] + s[11:3:-1] + s[2] + s[12]

elif len(s) == 87:

return s[62] + s[82:62:-1] + s[83] + s[61:52:-1] + s[0] + s[51:2:-1]

elif len(s) == 86:

return s[2:63] + s[82] + s[64:82] + s[63]

elif len(s) == 85:

return s[76] + s[82:76:-1] + s[83] + s[75:60:-1] + s[0] + s[59:50:-1] + s[1] + s[49:2:-1]

elif len(s) == 84:

return s[83:36:-1] + s[2] + s[35:26:-1] + s[3] + s[25:3:-1] + s[26]

elif len(s) == 83:

return s[6] + s[3:6] + s[33] + s[7:24] + s[0] + s[25:33] + s[53] + s[34:53] + s[24] + s[54:]

elif len(s) == 82:

return s[36] + s[79:67:-1] + s[81] + s[66:40:-1] + s[33] + s[39:36:-1] + s[40] + s[35] + s[0] + s[67] + s[32:0:-1] + s[34]

elif len(s) == 81:

return s[6] + s[3:6] + s[33] + s[7:24] + s[0] + s[25:33] + s[2] + s[34:53] + s[24] + s[54:81]

elif len(s) == 92:

return s[25] + s[3:25] + s[0] + s[26:42] + s[79] + s[43:79] + s[91] + s[80:83];

#else:

# print ('Unable to decrypt signature, key length %d not supported; retrying might work' % (len(s)))

def ExtractFlashVars(data, assets=False):

flashvars = {}

found = False

for line in data.split("\n"):

if line.strip().find(";ytplayer.config = ") > 0:

found = True

p1 = line.find(";ytplayer.config = ") + len(";ytplayer.config = ") - 1

p2 = line.rfind(";")

if p1 <= 0 or p2 <= 0:

continue

data = line[p1 + 1:p2]

break

data = RemoveAdditionalEndingDelimiter(data)

if found:

data = json.loads(data)

if assets:

flashvars = data['assets']

else:

flashvars = data['args']

return flashvars

def FetchPage(url):

req = urllib2.Request(url)

req.add\_header('User-Agent', 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.3) Gecko/2008092417 Firefox/3.0.3')

req.add\_header('Referer', 'http://www.youtube.com/')

return urllib2.urlopen(req).read().decode("utf-8")

def replaceHTMLCodes(txt):

# Fix missing ; in &#<number>;

txt = re.sub("(&#[0-9]+)([^;^0-9]+)", "\\1;\\2", txt)

txt = HTMLParser.HTMLParser().unescape(txt)

txt = txt.replace("&amp;", "&")

return txt

def RemoveAdditionalEndingDelimiter(data):

pos = data.find("};")

if pos != -1:

data = data[:pos + 1]

return data

####################################################

global playerData

global allLocalFunNamesTab

global allLocalVarNamesTab

def \_extractVarLocalFuns(match):

varName, objBody = match.groups()

output = ''

for func in objBody.split( '},' ):

output += re.sub(

r'^([^:]+):function\(([^)]\*)\)',

r'function %s\_\_\1(\2,\*args)' % varName,

func

) + '\n'

return output

def \_jsToPy(jsFunBody):

pythonFunBody = re.sub(r'var ([^=]+)={(.\*?)}};', \_extractVarLocalFuns, jsFunBody)

pythonFunBody = re.sub(r'function (\w\*)\$(\w\*)', r'function \1\_S\_\2', pythonFunBody)

pythonFunBody = pythonFunBody.replace('function', 'def').replace('{', ':\n\t').replace('}', '').replace(';', '\n\t').replace('var ', '')

pythonFunBody = pythonFunBody.replace('.reverse()', '[::-1]')

lines = pythonFunBody.split('\n')

for i in range(len(lines)):

# a.split("") -> list(a)

match = re.search('(\w+?)\.split\(""\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), 'list(' + match.group(1) + ')')

# a.length -> len(a)

match = re.search('(\w+?)\.length', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), 'len(' + match.group(1) + ')')

# a.slice(3) -> a[3:]

match = re.search('(\w+?)\.slice\((\w+?)\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), match.group(1) + ('[%s:]' % match.group(2)) )

# a.join("") -> "".join(a)

match = re.search('(\w+?)\.join\(("[^"]\*?")\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), match.group(2) + '.join(' + match.group(1) + ')' )

# a.splice(b,c) -> del a[b:c]

match = re.search('(\w+?)\.splice\(([^,]+),([^)]+)\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), 'del ' + match.group(1) + '[' + match.group(2) + ':' + match.group(3) + ']' )

pythonFunBody = "\n".join(lines)

pythonFunBody = re.sub(r'(\w+)\.(\w+)\(', r'\1\_\_\2(', pythonFunBody)

pythonFunBody = re.sub(r'([^=])(\w+)\[::-1\]', r'\1\2.reverse()', pythonFunBody)

return pythonFunBody

def \_jsToPy1(jsFunBody):

pythonFunBody = jsFunBody.replace('function', 'def').replace('{', ':\n\t').replace('}', '').replace(';', '\n\t').replace('var ', '')

pythonFunBody = pythonFunBody.replace('.reverse()', '[::-1]')

lines = pythonFunBody.split('\n')

for i in range(len(lines)):

# a.split("") -> list(a)

match = re.search('(\w+?)\.split\(""\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), 'list(' + match.group(1) + ')')

# a.length -> len(a)

match = re.search('(\w+?)\.length', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), 'len(' + match.group(1) + ')')

# a.slice(3) -> a[3:]

match = re.search('(\w+?)\.slice\(([0-9]+?)\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), match.group(1) + ('[%s:]' % match.group(2)) )

# a.join("") -> "".join(a)

match = re.search('(\w+?)\.join\(("[^"]\*?")\)', lines[i])

if match:

lines[i] = lines[i].replace( match.group(0), match.group(2) + '.join(' + match.group(1) + ')' )

return "\n".join(lines)

def \_getLocalFunBody(funName):

# get function body

funName = funName.replace('$', '\\$')

match = re.search('(function %s\([^)]+?\){[^}]+?})' % funName, playerData)

if match:

return match.group(1)

return ''

def \_getAllLocalSubFunNames(mainFunBody):

match = re.compile('[ =(,](\w+?)\([^)]\*?\)').findall( mainFunBody )

if len(match):

# first item is name of main function, so omit it

funNameTab = set( match[1:] )

return funNameTab

return set()

def \_extractLocalVarNames(mainFunBody):

valid\_funcs = ( 'reverse', 'split', 'splice', 'slice', 'join' )

match = re.compile( r'[; =(,](\w+)\.(\w+)\(' ).findall( mainFunBody )

local\_vars = []

for name in match:

if name[1] not in valid\_funcs:

local\_vars.append( name[0] )

return set(local\_vars)

def \_getLocalVarObjBody(varName):

match = re.search( r'var %s={.\*?}};' % varName, playerData )

if match:

return match.group(0)

return ''

def DecryptSignatureNew(s, playerUrl):

if not playerUrl.startswith('http:'):

playerUrl = 'http:' + playerUrl

#print "Decrypt\_signature sign\_len[%d] playerUrl[%s]" % (len(s), playerUrl)

global allLocalFunNamesTab

global allLocalVarNamesTab

global playerData

allLocalFunNamesTab = []

allLocalVarNamesTab = []

playerData = ''

request = urllib2.Request(playerUrl)

#res = core.\_fetchPage({u"link": playerUrl})

#playerData = res["content"]

try:

playerData = urllib2.urlopen(request).read()

playerData = playerData.decode('utf-8', 'ignore')

except Exception, e:

#print str(e)

print 'Failed to decode playerData'

return ''

# get main function name

match = re.search("signature=([$a-zA-Z]+)\([^)]\)", playerData)

if match:

mainFunName = match.group(1)

else:

print('Failed to get main signature function name')

return ''

\_mainFunName = mainFunName.replace('$','\_S\_')

fullAlgoCode = \_getfullAlgoCode(mainFunName)

# wrap all local algo function into one function extractedSignatureAlgo()

algoLines = fullAlgoCode.split('\n')

for i in range(len(algoLines)):

algoLines[i] = '\t' + algoLines[i]

fullAlgoCode = 'def extractedSignatureAlgo(param):'

fullAlgoCode += '\n'.join(algoLines)

fullAlgoCode += '\n\treturn %s(param)' % \_mainFunName

fullAlgoCode += '\noutSignature = extractedSignatureAlgo( inSignature )\n'

# after this function we should have all needed code in fullAlgoCode

#print '---------------------------------------'

#print '| ALGO FOR SIGNATURE DECRYPTION |'

#print '---------------------------------------'

#print fullAlgoCode

#print '---------------------------------------'

try:

algoCodeObj = compile(fullAlgoCode, '', 'exec')

except:

print 'Failed to obtain decryptSignature code'

return ''

# for security allow only flew python global function in algo code

vGlobals = {"\_\_builtins\_\_": None, 'len': len, 'list': list}

# local variable to pass encrypted sign and get decrypted sign

vLocals = { 'inSignature': s, 'outSignature': '' }

# execute prepared code

try:

exec(algoCodeObj, vGlobals, vLocals)

except:

print 'decryptSignature code failed to exceute correctly'

return ''

#print 'Decrypted signature = [%s]' % vLocals['outSignature']

return vLocals['outSignature']

# Note, this method is using a recursion

def \_getfullAlgoCode(mainFunName, recDepth=0):

global playerData

global allLocalFunNamesTab

global allLocalVarNamesTab

if MAX\_REC\_DEPTH <= recDepth:

print '\_getfullAlgoCode: Maximum recursion depth exceeded'

return

funBody = \_getLocalFunBody(mainFunName)

if funBody != '':

funNames = \_getAllLocalSubFunNames(funBody)

if len(funNames):

for funName in funNames:

funName\_ = funName.replace('$','\_S\_')

if funName not in allLocalFunNamesTab:

funBody=funBody.replace(funName,funName\_)

allLocalFunNamesTab.append(funName)

#print 'Add local function %s to known functions' % mainFunName

funbody = \_getfullAlgoCode(funName, recDepth+1) + "\n" + funBody

varNames = \_extractLocalVarNames(funBody)

if len(varNames):

for varName in varNames:

if varName not in allLocalVarNamesTab:

allLocalVarNamesTab.append(varName)

funBody = \_getLocalVarObjBody(varName) + "\n" + funBody

# convert code from javascript to python

funBody = \_jsToPy(funBody)

return '\n' + funBody + '\n'

return funBody