import base64

from struct import unpack, pack

import sys

import io

import os

import time

import itertools

import urllib2,urllib

import traceback

import urlparse

import posixpath

import re

import hmac

import hashlib

import binascii

import zlib

from hashlib import sha256, sha1,md5,sha512

import cookielib

import array

import socket

#from multiprocessing.connection import Client

from subprocess import Popen, PIPE

import subprocess

import struct

import pickle

import xbmc

import binascii

USEDec=0

initDone=False

def init():

# global initDone, androidClient,USEDec,popenProcess,authkey,portNumber

# if initDone: return

initDone=True

try:

from Crypto.Cipher import AES

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

print 'using pycrypt wooot woot'

except:

print 'pycrypt not available trying other options'

print traceback.print\_exc()

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

#check if its android

if xbmc.getCondVisibility('System.Platform.Android'):

try:

import androidsslPy

AES=androidsslPy.\_load\_crypto\_libcrypto()

USEDec=2 ## android

print 'using android ssllib woot woot'

except:

print traceback.print\_exc()

print 'android copy not available'

from f4mUtils import python\_aes

print 'using slow decryption'

else:

print 'using slow decryption'

from f4mUtils import python\_aes

value\_unsafe = '%+&;#'

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

if chr(c) not in value\_unsafe)

def tagDecrypt(data,key):

enc\_data=data#binascii.unhexlify(enc\_data)

enc\_key=key#binascii.unhexlify(enc\_key)

# print 'DataIn',binascii.hexlify(data)

# print 'KeyIn',binascii.hexlify(key)

keydatalen=0

if 'key\_' in enc\_data[0:300]: #quick check?? need to find better way to predict offsets

keydatalen=enc\_data[0:300].find(chr(0),13+16)-(13+16)+1

# print 'keydatalen',keydatalen

stage\_4a\_finaldataIndex=13+16+1+keydatalen #?? dynamic calc req

enc\_data\_index=stage\_4a\_finaldataIndex+32+40

stage\_4a\_finaldata=enc\_data[stage\_4a\_finaldataIndex:stage\_4a\_finaldataIndex+32]

globalivIndex=13

global\_iv=enc\_data[globalivIndex:globalivIndex+16]

# print 'global iv',binascii.hexlify(global\_iv)

stage\_4a\_data=enc\_key+global\_iv

# print len (stage\_4a\_data)

#??static data

stage\_4a\_key=binascii.unhexlify("3b27bdc9e00fd5995d60a1ee0aa057a9f1416ed085b21762110f1c2204ddf80ec8caab003070fd43baafdde27aeb3194ece5c1adff406a51185eb5dd7300c058")

# stage\_4a\_key=key#fixed

# print 'stage\_4a\_key',binascii.hexlify(stage\_4a\_key),len(stage\_4a\_key)

# print 'data',binascii.hexlify(stage\_4a\_data) ,len(stage\_4a\_data)

stage\_4a\_key2 = hmac.new(stage\_4a\_key,stage\_4a\_data , sha1).digest()

#stage\_4a\_key2+=chr(0)\*12

# print 'first HMAC ',binascii.hexlify(stage\_4a\_key2) ,len(stage\_4a\_key2)

#??static data

stage\_4a\_data2=binascii.unhexlify("d1ba6371c56ce6b498f1718228b0aa112f24a47bcad757a1d0b3f4c2b8bd637cb8080d9c8e7855b36a85722a60552a6c00")

# print 'stage\_4a\_data2',binascii.hexlify(stage\_4a\_data2),len(stage\_4a\_data2)

auth = hmac.new(stage\_4a\_key2,stage\_4a\_data2 , sha1).digest()

stage\_4a\_finalkey=auth[:16]

# print stage\_4a\_finalkey, repr(stage\_4a\_finalkey), len(stage\_4a\_finalkey)

# print binascii.hexlify(stage\_4a\_finalkey)

# print 'first end HMAC >>>>>>>>>>>>>>>>>>>>>>>>>'

# print 'final data',binascii.hexlify(stage\_4a\_finaldata)

# print 'final iv',binascii.hexlify(global\_iv)

# print 'final key',binascii.hexlify(stage\_4a\_finalkey)

#import pyaes

# de =AES.new(stage\_4a\_finalkey, AES.MODE\_CBC, global\_iv)

# # pyaes.new(stage\_4a\_finalkey, pyaes.MODE\_CBC, IV=global\_iv)

de=getDecrypter(stage\_4a\_finalkey,global\_iv )

cc=global\_iv

decresp=decryptData(de,stage\_4a\_finaldata,cc)

stage\_4a\_finaloutput=decresp[:20]

enc\_size=stage\_4a\_finaloutput[:4]

enc\_size=int(struct.unpack('>I',enc\_size)[0])

# print stage\_4a\_finaloutput

stage\_4a\_finaloutput=stage\_4a\_finaloutput[4:4+16]

# print 'final',binascii.hexlify(stage\_4a\_finaloutput)

stage\_4\_key=stage\_4a\_key

stage\_5\_key = hmac.new(stage\_4\_key,stage\_4a\_finaloutput , sha1).digest()

# print 'stage\_4\_hmac ',binascii.hexlify(stage\_5\_key)

#??static data

stage\_5\_data=binascii.unhexlify("d1ba6371c56ce6b498f1718228b0aa112f24a47bcad757a1d0b3f4c2b8bd637cb8080d9c8e7855b36a85722a60552a6c01")

# print 'stage\_5\_data',binascii.hexlify(stage\_5\_data),len(stage\_5\_data)

stage\_5\_hmac = hmac.new(stage\_5\_key,stage\_5\_data , sha1).digest()

# print 'stage\_5\_hmac ',binascii.hexlify(stage\_5\_hmac), len(stage\_5\_hmac)

stage\_5\_hmac=stage\_5\_hmac[:16]

# print 'stage\_5\_hmac trmimed ',binascii.hexlify(stage\_5\_hmac), len(stage\_5\_hmac)

# de =AES.new(stage\_5\_hmac, AES.MODE\_CBC, global\_iv)

# #de = pyaes.new(stage\_5\_hmac, pyaes.MODE\_CBC, IV=global\_iv)

de2=getDecrypter(stage\_5\_hmac,global\_iv )

enc\_data\_todec=""

if enc\_size>0:

enc\_data\_todec=enc\_data[enc\_data\_index:enc\_data\_index+enc\_size]

unEncdata=enc\_data[enc\_data\_index+enc\_size:]

decData=""

if len(enc\_data\_todec)>0:

# print 'enc\_data\_todec',binascii.hexlify(enc\_data\_todec), len(enc\_data\_todec)

#enc\_data\_remaining

decData=decryptData(de2,enc\_data\_todec,global\_iv)

decData+=unEncdata

if 1==2 and len(decData)<300:

print 'enc data received',binascii.hexlify(enc\_data\_todec), len(enc\_data\_todec)

print 'iv received',binascii.hexlify(global\_iv), len(global\_iv)

print 'key received',binascii.hexlify(stage\_5\_hmac), len(stage\_5\_hmac)

print 'data received',binascii.hexlify(data), len(data)

print 'final return',binascii.hexlify(decData), len(decData)

return decData

## function to create the cbc decrypter object

def getDecrypter(key,iv):

global USEDec

if USEDec==1:

enc =AES.new(key, AES.MODE\_CBC, iv)

elif USEDec==3:

ivb=array.array('B',iv)

keyb= array.array('B',key)

enc=python\_aes.new(keyb, 2, ivb)

else:

enc =androidsslPy.\_load\_crypto\_libcrypto()

enc = enc(key, iv)

return enc

## function to create the cbc decrypter

def decryptData(d,encdata,iv):

# print 'start'

global USEDec

if USEDec==1 or USEDec==2:

data =d.decrypt(encdata)

# print binascii.hexlify(data)

elif USEDec==3:

chunkb=array.array('B',encdata)

data = d.decrypt(chunkb)

data="".join(map(chr, data))

# print 'end'

return data

def cleanup():

try:

if USEDec==2:

print 'doing android cleanup'

#AndroidCrypto.teardown()

#print 'android cleanup'

except:

pass

#enc\_data=""

#byte[] arrOutput = { 0x0C, 0x00, 0x00, 0x00, 0x00, 0x55, 0xEA, 0x65, 0x7E, 0x00, 0x00, 0xFF, 0xFF, 0xE9, 0x86, 0x40, 0x1A, 0x2B, 0xDB, 0x60, 0x36, 0xEC, 0x24, 0xB3, 0x47, 0xA3, 0xF4, 0x91, 0x40, 0x2F, 0x7A, 0x2F, 0x41, 0x45, 0x54, 0x4E, 0x2D, 0x48, 0x69, 0x73, 0x74, 0x6F, 0x72, 0x79, 0x5F, 0x56, 0x4D, 0x53, 0x2F, 0x42, 0x52, 0x41, 0x4E, 0x44, 0x5F, 0x54, 0x48, 0x43, 0x5F, 0x4F, 0x43, 0x48, 0x41, 0x5F, 0x31, 0x37, 0x36, 0x38, 0x34, 0x37, 0x5F, 0x53, 0x46, 0x4D, 0x5F, 0x30, 0x30, 0x30, 0x5F, 0x32, 0x39, 0x39, 0x37, 0x5F, 0x31, 0x35, 0x5F, 0x32, 0x30, 0x31, 0x35, 0x30, 0x39, 0x30, 0x31, 0x5F, 0x30, 0x30, 0x5F, 0x53, 0x33, 0x5F, 0x2C, 0x34, 0x2C, 0x31, 0x38, 0x2C, 0x31, 0x33, 0x2C, 0x31, 0x30, 0x2C, 0x37, 0x2C, 0x32, 0x2C, 0x31, 0x2C, 0x30, 0x30, 0x2E, 0x6D, 0x70, 0x34, 0x2E, 0x63, 0x73, 0x6D, 0x69, 0x6C, 0x2F, 0x6B, 0x65, 0x79, 0x5F, 0x41, 0x51, 0x42, 0x63, 0x41, 0x54, 0x31, 0x51, 0x50, 0x68, 0x69, 0x44, 0x6B, 0x33, 0x35, 0x6C, 0x36, 0x6C, 0x56, 0x39, 0x56, 0x62, 0x54, 0x7A, 0x35, 0x56, 0x4C, 0x32, 0x31, 0x56, 0x61, 0x71, 0x76, 0x4F, 0x41, 0x38, 0x42, 0x79, 0x79, 0x32, 0x47, 0x31, 0x35, 0x68, 0x4C, 0x4D, 0x34, 0x65, 0x58, 0x48, 0x63, 0x54, 0x6D, 0x67, 0x70, 0x67, 0x7A, 0x67, 0x43, 0x5A, 0x34, 0x77, 0x44, 0x50, 0x55, 0x62, 0x6A, 0x48, 0x44, 0x5A, 0x31, 0x6E, 0x00, 0x01, 0xEF, 0x4E, 0x24, 0x89, 0x4B, 0x22, 0x08, 0x93, 0xDD, 0xD2, 0x9D, 0xA3, 0xD6, 0xBA, 0x3A, 0xB4, 0x98, 0x57, 0x84, 0x80, 0x7E, 0xD9, 0xA7, 0xA4, 0x49, 0x85, 0xCF, 0xA2, 0xAD, 0x4B, 0x22, 0xD9, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x38, 0x73, 0xD0, 0xAB, 0x9F, 0xB7, 0x4C, 0x81, 0xEE, 0xCD, 0xD2, 0xB7, 0x87, 0xE2, 0x98, 0x96, 0xE8, 0x8A, 0x98, 0xC6, 0x4E, 0x79, 0xC9, 0x53, 0x3C, 0x8F, 0xDA, 0xDE, 0xEE, 0xF7, 0x84, 0x16, 0xC8, 0x4F, 0x75, 0xB4, 0x7B, 0x7C, 0xF8, 0x61, 0xA7, 0x2B, 0x54, 0xF3, 0x06, 0xD5, 0x3F, 0xEE, 0xDF, 0xF2, 0xD1, 0x60, 0x8F, 0x18, 0x32, 0x58, 0x01, 0x00, 0x05, 0x68, 0xE9, 0x33, 0x2C, 0x80, 0x00, 0x00, 0x01, 0x53, 0x0A, 0x00, 0x00, 0x6D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x55, 0xEA, 0x65, 0x7E, 0x00, 0x00, 0xFF, 0xFB, 0xE9, 0x86, 0x40, 0x1A, 0x2B, 0xDB, 0x60, 0x36, 0xEC, 0x24, 0xB3, 0x47, 0xA3, 0xF4, 0x91, 0x40, 0x01, 0x7B, 0x4C, 0x44, 0x02, 0x0B, 0xD1, 0xBD, 0x24, 0xD1, 0x15, 0x35, 0xAA, 0x24, 0x67, 0x1B, 0x89, 0xA7, 0x70, 0xB7, 0xF6, 0xF5, 0x62, 0x1E, 0xCB, 0xDA, 0x7B, 0x44, 0x77, 0x29, 0xF0, 0x42, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xCE, 0x2E, 0x6C, 0x71, 0x3B, 0x79, 0x1D, 0x96, 0x0C, 0x49, 0x83, 0x94, 0x9A, 0xD0, 0x25, 0x0D, 0x3B, 0xD2, 0x68, 0xB7, 0xAF, 0x00, 0x11, 0x90, 0x56, 0xE5, 0x00, 0x00, 0x00, 0x00, 0x78, 0x0B, 0x00, 0x6D, 0xE3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x55, 0xEA, 0x65, 0x7E, 0x00, 0x00, 0xFF, 0xFB, 0xE9, 0x86, 0x40, 0x1A, 0x2B, 0xDB, 0x60, 0x36, 0xEC, 0x24, 0xB3, 0x47, 0xA3, 0xF4 };

#enc\_data=binascii.unhexlify(enc\_data)

#enc\_key="93ac1d5925eadd38f61fee4c321cc843"

#enc\_key=binascii.unhexlify(enc\_key)

#print 'final data',binascii.hexlify(tagDecrypt(enc\_data,enc\_key) )