基于Xposed Hook实现的Android App的协议算法分析小工具-CryptoFucker

原创

Fly20141201 于 2018-07-08 20:41:35 发布 6516 收藏 12
分类专栏： Android逆向学习 Android系统安全和逆向分析研究 文章标签： xposed hook CryptoFucker

本文链接：https://blog.csdn.net/QQ1084283172/article/details/80962121


Android逆向学习 同时被 2 个专栏收录
58 篇文章 6 订阅
订阅专栏


Android系统安全和逆向分析研究
72 篇文章 59 订阅
订阅专栏

在进行Android应用的网络协议分析的时候，不可避免涉及到网络传输数据的加密算法的分析，这里分享一下作者无名侠写的一个小工具 CryptoFucker，看雪论坛的原帖子《[推荐]【Tools】CryptoFucker》.

CryptoFucker工具的github地址:https://github.com/Chenyuxin/CryptoFucker

CryptoFucker工具的使用说明:

CryptoFucker

Xposed框架，用于抓取javax.crypto. *与javax.security.* 算法参数（包括加密数据、密钥、IV、结果等数据）的工具。

使用方法

- 1.安装Xposed framework
- 2.安装并激活CryptoFucker
- 3.运行你想测试的APP
- 4./sdcard/ydsec/packgeName.txt 为数据文件

建议使用Notepad++查看数据文件。

优势

- 支持 javax.crypto.* 大部分函数
- 支持 javax.security.*大部分函数
- IV向量嗅探
- 密钥嗅探
- 加密原始数据嗅探
- 加密结果嗅探
- 调用栈显示
- HEX 显示

github:https://github.com/Chenyuxin/CryptoFucker

Xposed Hook的结果日志存放路径为 **/sdcard/ydsec/packgeName.txt**，日志文件的格式如下图所示：

```
AES Key

0x00000000 43 57 45 4C 57 52 52 45 57 34 35 36 37 69 31 6F
--------------------------------------------------------------------------------

Iv

0x00000000 43 57 45 4C 57 52 52 45 57 34 35 36 37 69 31 6F
--------------------------------------------------------------------------------

AES/CBC/NoPadding Data:

0x00000000 7B 22 68 65 6C 70 65 72 5F 76 65 72 73 69 6F 6E {"helper_version
0x00000010 22 3A 22 35 2E 30 2E 30 2E 37 2E 51 22 2C 22 67 ":"5.0.0.7.Q","g
0x00000020 61 69 64 22 3A 22 34 36 32 34 63 34 39 63 2D 61 aid":"4624c49c-a
0x00000030 32 39 62 2D 34 62 36 65 2D 61 62 38 65 2D 66 32 29b-4b6e-ab8e-f2
0x00000040 62 32 39 39 33 61 66 37 38 65 22 2C 22 70 6B 67 b2993af78e","pkg
0x00000050 5F 6E 61 6D 65 22 3A 22 63 6F 6D 2E 65 78 61 6D _name":"com.exam
0x00000060 70 6C 65 2E 64 65 6D 6F 22 2C 22 61 70 6B 5F 76 ple.demo","apk_v
0x00000070 65 72 73 69 6F 6E 22 3A 22 33 2E 31 30 2E 31 38 ersion":"3.10.18
0x00000080 5F 77 77 22 2C 22 61 69 64 22 3A 22 66 63 66 31 _ww","aid":"fcf1
0x00000090 66 61 66 37 36 61 35 64 61 34 38 36 22 2C 22 6C faf76a5da486","l
0x000000A0 61 6E 67 75 61 67 65 22 3A 22 7A 68 22 2C 22 61 anguage":"zh","a
0x000000B0 70 70 5F 6E 61 6D 65 22 3A 22 63 6F 6D 2E 6C 65 pp_name":"com.le
0x000000C0 6E 6F 76 6F 2E 61 6E 79 73 68 61 72 65 2E 67 70 novo.anyshare.gp
0x000000D0 73 22 2C 22 63 6F 75 6E 74 72 79 22 3A 22 43 4E s","country":"CN
0x000000E0 22 2C 22 6F 73 5F 76 65 72 73 69 6F 6E 22 3A 22 ","os_version":"
0x000000F0 34 2E 34 2E 34 22 7D 00 00 00 00 00 00 00 00 00
--------------------------------------------------------------------------------

AES/CBC/NoPadding  result:

0x00000000 6A 11 DC D9 C5 F5 38 E6 04 8C 3B 74 A8 5A C3 EC j.....8...;t.Z..
0x00000010 DF 5A 64 41 27 27 9D 9D 58 05 63 F1 D6 20 25 97 .ZdA''..X.c... %.
0x00000020 1E 53 22 E3 9E 70 D5 B0 C5 A4 15 37 D1 A5 44 25 .S"..p.....7..D%
0x00000030 FB 18 19 75 A2 44 20 6B 6C BC 8E A9 6C 82 CD 0A ...u.D.kl...l...
0x00000040 8E 49 F1 3C 59 8C BC 58 74 B7 54 78 B4 CB 6D 4A .I.<Y..Xt.Tx..mJ
0x00000050 AF 09 D2 EF 7C 13 0D 0D FB 75 8A A6 05 4B 92 39 ....|....u...K.9
0x00000060 48 90 7F BA C7 2C 42 00 1A 3A 44 A9 67 F3 56 05 H....,B..:D.g.V.
0x00000070 14 92 7E 00 66 76 33 CD 9D 87 9B 26 91 4A 65 32 ....fv3....&.Je2
0x00000080 D3 8D 6B 37 FE 8E 3C 4D 98 2B 00 87 54 C2 84 4A ..k7..<M.+..T..J
0x00000090 7F F9 91 F2 6B A0 F9 D1 CC 3D 61 45 B2 22 28 CA ....k....=aE."(.
0x000000A0 FC 73 B7 91 32 C8 FE 36 FC 4C 36 DA 83 C9 CB 11 .s..2..6.L6.....
```

关键代码 **TestHook.java** 的注释和学习：

```
package com.example.a14473.xp;

import android.os.Bundle;
import android.support.annotation.NonNull;
import android.util.Log;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Array;
import java.util.HashMap;
import java.nio.ByteBuffer;
import java.security.Key;
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.security.cert.Certificate;
import java.security.spec.KeySpec;
import java.util.Map.Entry;
import java.util.HashMap;
import java.util.Objects;
```

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.DESKeySpec;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XposedBridge;
import de.robv.android.xposed.XposedHelpers;
import de.robv.android.xposed.callbacks.XC_LoadPackage;

import static de.robv.android.xposed.XposedHelpers.findAndHookConstructor;
import static de.robv.android.xposed.XposedHelpers.findAndHookMethod;

import com.example.a14473.xp.HexDumper;

/**
 * Created by 14473 on 2017/7/2.
 */
public class TestHook implements IXposedHookLoadPackage {

    @Override
    public void handleLoadPackage(final XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable

     // 打印当前Android应用的进程的名字和包名
        String logstr  = " W:" + loadPackageParam.processName + "-"+loadPackageParam.packageName;
        XposedBridge.log(logstr);
        // 这里其实可以增加一下指定名称的Android应用程序的包名的过滤

        try {

         // 先调用函数XposedHelpers.findClass进行类的加载处理
         // java Hook处理类javax.crypto.spec.DESKeySpec的所有构造函数
            XposedBridge.hookAllConstructors(XposedHelpers.findClass("javax.crypto.spec.DESKeySpec", loadPa
              new XC_MethodHook() {

                @Override
                protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                    super.beforeHookedMethod(param);

                    String keystr;
                    // 申请内存空间存放数据
                    byte[] keybyte = new byte[8];
                    int offset = 0;
                    // 如果有两个参数的构造函数，第二个参数是偏移
                    // param.args.length获取函数的传入参数的个数
                    if(param.args.length != 1)
                        offset = (int)param.args[1];
                    // 拷贝数据到申请的内存空间中
                    System.arraycopy((byte[])param.args[0], offset, keybyte, 0, 8);

                    // log日志文件中前置tag
                    keystr = "DES KEY";
                    // 打印数据到指定的log日志文件中
                    Util.MyLog(loadPackageParam.packageName, keystr, keybyte);
                }
            });
```

```
});

// java Hook处理类"javax.crypto.spec.DESedeKeySpec"的所有构造函数
XposedBridge.hookAllConstructors(XposedHelpers.findClass("javax.crypto.spec.DESedeKeySpec", loa
  new XC_MethodHook() {

    @Override
    protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
        super.beforeHookedMethod(param);

        String keystr;
        // 申请内存空间存放数据
        byte[] keybyte = new byte[24];
        int offset = 0;

        // 如果有两个参数的构造函数，第二个参数是偏移
        if(param.args.length != 1)
            offset = (int)param.args[1];
        // 拷贝数据到申请的内存空间中
        System.arraycopy((byte[])param.args[0], offset, keybyte, 0, 24);

        // log日志文件中前置tag
        keystr = "3DES KEY";
        // 打印数据到指定的log日志文件中
        Util.MyLog(loadPackageParam.packageName,keystr,keybyte);
    }
});

// java Hook处理类"javax.crypto.spec.SecretKeySpec"的所有构造函数
XposedBridge.hookAllConstructors(XposedHelpers.findClass("javax.crypto.spec.SecretKeySpec", loa
  new XC_MethodHook() {

    @Override
    protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
        super.beforeHookedMethod(param);

        int offset = 0;
        int size = 0;
        String Algorithm;

        if(param.args.length != 2)
        {
            offset = (int)param.args[1];
            size = (int)param.args[2];
            Algorithm = (String)param.args[3];
        }else {
            Algorithm = (String) param.args[1];
            size = ((byte[])param.args[0]).length;
        }

        byte[] data = new byte[size];
        System.arraycopy((byte[])param.args[0], offset, data, 0, size);

        String str ;
        // log日志文件中前置tag
        str = Algorithm + " Key";
        Util.MyLog(loadPackageParam.packageName,str,data);
    }
});
```

```
        // IV 向量
        // java Hook处理类"javax.crypto.spec.IvParameterSpec"的所有构造函数
        XposedBridge.hookAllConstructors(XposedHelpers.findClass("javax.crypto.spec.IvParameterSpec", l
          new XC_MethodHook() {

            @Override
            protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                super.beforeHookedMethod(param);

                String keystr;
                byte[] IVByte;
                byte[] tmp;
                int offset = 0;
                int size;
                tmp = (byte[])param.args[0];
                size = tmp.length;

                // 如果有两个参数的构造函数，第二个参数是偏移
                if(param.args.length != 1)
                {
                    offset = (int)param.args[1];
                    size = (int)param.args[2];
                }
                IVByte = new byte[size];
                System.arraycopy(tmp, offset, IVByte, 0, size);

                // log日志文件中前置tag
                keystr = "Iv";
                Util.MyLog(loadPackageParam.packageName,keystr,IVByte);
            }
        });

        // java Hook处理类"javax.crypto.Cipher"中所有名称为"doFinal"的类方法
        XposedBridge.hookAllMethods(XposedHelpers.findClass("javax.crypto.Cipher", loadPackageParam.cla
                "doFinal", new XC_MethodHook() {

                @Override
                protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                    super.afterHookedMethod(param);

                    Cipher cip = (Cipher)param.thisObject;
                    if(param.args.length >= 1)
                    {
                        // log日志文件中前置tag
                        String str = cip.getAlgorithm() + " Data:";
                        Util.MyLog(loadPackageParam.packageName,str,(byte[])param.args[0]);

                        // log日志文件中前置tag
                        str = cip.getAlgorithm() + "  result:";
                        Util.MyLog(loadPackageParam.packageName, str, (byte[])param.getResult());
                    }
                }
        });

        // java Hook处理类"java.security.MessageDigest"中所有名称为"update"的类方法
        XposedBridge.hookAllMethods(XposedHelpers.findClass("java.security.MessageDigest", loadPackageP
          new XC_MethodHook() {

            @Override
            protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
```

```
                protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                    super.beforeHookedMethod(param);

                    MessageDigest md = (MessageDigest)param.thisObject;
                    String str = md.getAlgorithm() + " update data:";
                    Util.MyLog(loadPackageParam.packageName, str, (byte[])param.args[0]);
                }
            });

            // java Hook处理类"java.security.MessageDigest"中所有名称"digest"的类方法
            XposedBridge.hookAllMethods(XposedHelpers.findClass("java.security.MessageDigest", loadPackageP
              new XC_MethodHook() {

                @Override
                protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                    super.afterHookedMethod(param);
                    if (param.args.length >= 1)
                    {
                        MessageDigest md = (MessageDigest)param.thisObject;

                        String str;
                        str = md.getAlgorithm() + "  data:";
                        Util.MyLog(loadPackageParam.packageName,str,(byte[])param.args[0]);

                        str = md.getAlgorithm() + "  result:";
                        Util.MyLog(loadPackageParam.packageName,str,(byte[])param.getResult());
                    }
                }
            });
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}


class Util {

 // 将字节数组数据转换为16进制的大写字符串
    @NonNull
    public static String byteArrayToString(byte[] bytes) {

        String hs = "";
        String tmp = "";
        for (int n = 0; n < bytes.length; n++) {

            //整数转成十六进制表示
            tmp = (java.lang.Integer.toHexString(bytes[n] & 0XFF));
            if (tmp.length() == 1) {

                hs = hs + "0" + tmp;
            } else {

                hs = hs + tmp;
            }
        }
        tmp = null;

        //转成大写
        return hs.toUpperCase();
```

```
    }

    // 获取类方法堆栈调用的信息字符串
    public static String GetStack()
    {
        String result = "";
        Throwable ex = new Throwable();
        StackTraceElement[] stackElements = ex.getStackTrace();
        if (stackElements != null) {

            int range_start = 3;
            int range_end = Math.min(stackElements.length, 7);
            if(range_end < range_start)
                return  "";
            // 获取合理调用层次的堆栈信息
            for (int i = range_start; i < range_end; i++) {

                result = result + (stackElements[i].getClassName()+"->");  // 类的名称
                result = result + (stackElements[i].getMethodName())+"  "; // 类方法的名称
                result = result + (stackElements[i].getFileName()+"(");  // 源码文件的名称
                result = result + (stackElements[i].getLineNumber()+")\n"); // 源码在文件中的行号
                result = result + ("----------------------------------\n");
            }
        }
        return result;
    }

    // 打印的log日志文件的名称格式为/sdcard/ydsec/packgeName.txt
    // ++++++ 注意是否有对sdcard文件目录的写操作权限 ++++++++++++
    public  static void MyLog(String packname, String info, byte[] data)
    {
     // 创建文件目录"/sdcard/ydsec/"
        String path = "/sdcard/ydsec/";
        File pather = new File(path);
        if(!pather.exists())
             pather.mkdir();
        // 拼接字符串得到log日志文件的路径
        String filename = path + packname+".txt";
        if(data.length >= 256)
            return;

        try
        {
         // 1.前置tag显示字符串
            info = info + "\n";
            // 2.类方法调用堆栈的信息
            info = info + GetStack() + "\n";
            // 3.需要打印的关键加密算法的数据信息
            info = info + HexDumper.dumpHexString(data) + "\n------------------------------------------------

            // 创建保存Log日志的文件/sdcard/ydsec/packgeName.txt
            FileWriter fw = new FileWriter(filename, true);
            // 将需要保存的数据信息写入到Log日志文件/sdcard/ydsec/packgeName.txt中
            fw.write(info);
            fw.close();

            // 打印log日志
            Log.d("q_"+packname,info);
            XposedBridge.log("["+ packname+"]"+info);
```

```
        } catch(IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

格式化打印Log日志的代码文件 **HexDumper.java** 的注释和学习：

```
package com.example.a14473.xp;

public class HexDumper
{

    private final static char[] HEX_DIGITS = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
     'A', 'B', 'C', 'D', 'E', 'F' };

    // 将字节数组的数据转换为16进制的字符串数据
    public static String dumpHexString(byte[] array)
    {
        if (array.equals(null))
            return "null";
        // 申请内存空间
        byte[] byte2 = new byte[array.length + 0x10];
        // 内存空间清零
        for(int i = 0; i < byte2.length; i++)
        {
            byte2[i] = 0;
        }

        // 将传入的字节数组中的数据拷贝到新数组中
        for(int i = 0; i < array.length; i++)
        {
            byte2[i] = array[i];
        }

        // 将字节数组数据转换为16进制的字符串数据
        return dumpHexString(byte2, 0, byte2.length);
    }

    // 将字节数组数据转换为16进制的字符串数据
    public static String dumpHexString(byte[] array, int offset, int length)
    {
        StringBuilder result = new StringBuilder();
        byte[] line = new byte[16];
        int lineIndex = 0;

        // 打印数据的偏移
        result.append("\n0x");
        // 将整型字节数组偏移转换为16进制字符串数据
        result.append(toHexString(offset));

        for (int i = offset; i < offset + length; i++)
        {
            if (lineIndex == 16)
            {
                result.append(" ");
```

```
            for (int j = 0 ; j < 16 ; j++)
            {
                if (line[j] > ' ' && line[j] < '~')
                {
                    result.append(new String(line, j, 1));
                }
                else
                {
                    result.append(".");
                }
            }
            // 打印数据的偏移
            result.append("\n0x");
            result.append(toHexString(i));
            lineIndex = 0;
        }

        byte b = array[i];
        result.append(" ");
        result.append(HEX_DIGITS[(b >>> 4) & 0x0F]);
        result.append(HEX_DIGITS[b & 0x0F]);

        line[lineIndex++] = b;
    }

    if (lineIndex != 16)
    {
        int count = (16 - lineIndex) * 3;
        count++;
        for (int i = 0 ; i < count ; i++)
        {
            result.append(" ");
        }

        for (int i = 0 ; i < lineIndex ; i++)
        {
            if (line[i] > ' ' && line[i] < '~')
            {
                result.append(new String(line, i, 1));
            }
            else
            {
                result.append(".");
            }
        }
    }
    return result.toString();
}

// 将单字节数组转化为16进制的字符串
public static String toHexString(byte b)
{
 // 将字节数组转换为16进制的字符串数据
    return toHexString(toByteArray(b));
}

// 将字节数组转换为16进制的字符串数据
public static String toHexString(byte[] array)
{
```

```
{
 // 将指定字节数组的指定偏移位置指定长度的字节数据转换为16进制字符串进行显示
    return toHexString(array, 0, array.length);
}

// 将指定字节数组的指定偏移位置指定长度的字节数据转换为16进制字符串进行显示
public static String toHexString(byte[] array, int offset, int length)
{
 // 申请内存空间存放字符数组
    char[] buf = new char[length * 2];
    int bufIndex = 0;

    for (int i = offset ; i < offset + length; i++)
    {
     // 取传入数组中的1字节数据
        byte b = array[i];
        // 取字节数据中高4位的数据转换为16进制字符串
        buf[bufIndex++] = HEX_DIGITS[(b >>> 4) & 0x0F];
        // 取字节数据中低4位的数据转换为16进制字符串
        buf[bufIndex++] = HEX_DIGITS[b & 0x0F];
    }
    // 返回最终转换成功的字符串
    return new String(buf);
}

// 将int整型数组转换为16进制的字符串进行显示
public static String toHexString(int i)
{
    // 将字节数组转换为16进制的字符串数据
    return toHexString(toByteArray(i));
}

// 将单字节数组转化为字节数组进行存储
public static byte[] toByteArray(byte b)
{
    byte[] array = new byte[1];
    array[0] = b;

    return array;
}

// 将int整型数转换为4字节的字节数组
public static byte[] toByteArray(int i)
{
    byte[] array = new byte[4];
    array[3] = (byte)(i & 0xFF);
    array[2] = (byte)((i >> 8) & 0xFF);
    array[1] = (byte)((i >> 16) & 0xFF);
    array[0] = (byte)((i >> 24) & 0xFF);

    return array;
}

// 将单个字符转换为整型
private static int toByte(char c)
{
    if (c >= '0' && c <= '9') return (c - '0');
    if (c >= 'A' && c <= 'F') return (c - 'A' + 10);
    if (c >= 'a' && c <= 'f') return (c - 'a' + 10);
```

```
        throw new RuntimeException ("Invalid hex char '" + c + "'");
    }

    // 将字符串数据转换为相应的字节数组数据进行存储
    public static byte[] hexStringToByteArray(String hexString)
    {
        int length = hexString.length();
        byte[] buffer = new byte[length / 2];
        // 每次处理2个字符的字符串
        for (int i = 0 ; i < length ; i += 2)
        {
         // 例如将"23"转换为0x23
            buffer[i / 2] = (byte)((toByte(hexString.charAt(i)) << 4) | toByte(hexString.charAt(i+1)));
        }
        return buffer;
    }
}
```

后面我会再写个类似的简单小工具~