[原创]梆梆企业版加固技术之防篡改剖析 优

oooAooo 5 9

学者

2017-12-6 00:18 10468

（本文仅限于技术讨论，不得用于非法途径，造成不良后果，与作者无关）

本篇主要介绍梆梆安全加固防篡改的原理和方法。选择梆梆，是因为其在防篡改方面是比较突出的。如果后续有时间会陆续将梆梆企业版使用的加固技术和大家分享。

1. 前言

既然我们选择BB，我们先看一下BB在防篡改技术的介绍，下图是BB官网上关于防篡改的介绍。（https://www.bangcle.com/products/productindex?product_id=1）

核心加固技术:

防逆向（Anti-RE） 抽取classes.dex中的所有代码，剥离敏感函数功能，混淆关键逻辑代码，整体文件深度加密加壳，防止通过apktool，dex2jar，JEB等静态工具来查看应用的Java层代码，防止通过IDA，readelf等工具对so里面的逻辑进行分析，保护native代码。

防篡改（Anti-tamper） 每个代码文件、资源文件、配置文件都会分配唯一识别指纹，替换任何一个文件，将会导致应用无法运行，存档替换、病毒广告植入、内购破解、功能屏蔽等恶意行为将无法实施。

防调试（Anti-debug） 多重加密技术防止代码注入，彻底屏蔽游戏外挂、应用辅助工具，避免钓鱼攻击、交易劫持、数据修改等调试行为。

防窃取（Storage Encryption） 支持存储数据加密，提供输入键盘保护、通讯协议加密、内存数据变换、异常进程动态跟踪等安防技术，有效防止针对应用动、静态数据的捕获、劫持和篡改。

从上图可知，APK“任何文件”（这里的任何加引号的意思是APKI里面还是有几个文件不在其防篡改的保护之内的，我们后面会说明）被替换或修改，应用都无法运行。实现的方法是文件都会分配一个识别指纹。

根据BB这个说明，我们将开始逆向分析其防篡改技术。实际在逆向分析中我们会发现其采用很多加密算法和数据结构。为了便于大家对了解其防篡改的技术全貌，先介绍分析完后的成果，然后在下一篇介绍这些成果是怎样分析过程。

2. 算法与数据结构

2.1 使用了如下6种加密算法：

- MD5
- RC4
- 斐波那契数列
- BAS64
- SHA-1
- RSA

2.2 使用了如下3种数据结构：

- UTHASH
- 单项链表
- ZIP

参考资料的2和3是关于UTHASH与sha1withRDA的介绍，感兴趣的话可以看一下。

2.3 自定义的结构变量

2.3.1 APK\assets\meta-data\ manifest.mf文件格式

BB在其防篡改功能说明中说到，会为每个文件分配一个识别指纹，那么这个指纹到底存在哪里呢。经过分析发现APK\assets\meta-data\ manifest.mf文件存放的就是对应的指纹信息。下面介绍一下这个文件的格式。

```
j20KiwhQW5kcm9pZE1hbmlmZXN0LnhtbA==sAdB/Mv
SOkYwu0prp/szv4eb6aG3PBhZKsWqkWBKaV9846zox
ff8tXFZ4eMjm8pFXXvD869etVeBlXtHx7dtYC27762
UXM5708mUVbAkiMfRLjIBsyeeaa8bZBhMKHUAv6DUJ
0RsmllQ8f94gYjYLf+lxePOBt5D9c07Oc3gdqGOmH5
icL3QJnWvjA4eb9+/aEORLJw9HWLSSCe287YJXl16s
wYkq8p7Q3NiOzlx7265DgthtPOycz+j8hyc4ec2pQI
dl88rUFVxlvTPLSQIVx5e2ciqYIaMPIg/dFDorie77
NnC4a9eJn4RrlxO/CGXRoifba6a8vLr275hV044ss8
```

图 2-1 [APK\assets\meta-data\ manifest.mf]

从上图中可以看到这个文件的内容看起来像是BASE64编码，实际上是不是呢，我们可以验证一下，先将文件开始像base64编码复制出来如下：

“j20KiwhQW5kcm9pZE1hbmlmZXN0LnhtbA==”但是这个字符串是35个字符，对于base64编码一定是4字节对齐的，不够则用=补齐。我们可以把前面的3个字符去掉，让其是32个字节（为啥去掉3个字节，实际上在后面逆向分析过程中我们可以明白），经过base64解码后，如下：

图 2-2 [APK\assets\meta-data\ manifest.mf 内容解码]

表 2-1 [[APK\assets\meta-data\ manifest.mf文件格式]

偏移	字节数	说明
0	1	其与 0x68 异或后为 2，表示后面字符串的长度
1	2	表示后面文件名对应的 base64 的长度
2	20	AndroidManifest.xml 字符串的 base64
36	28	文件指纹 3，原始 APK 的所有文件 sha1 的前 4 个字符组合在一起的 sha1 再进行 base64
64	28	实际分析中，并没有用到
92	28	实际分析中，并没有用到
120	28	实际分析中，并没有用到
148	28	文件指纹 4，原始 APK 所有文件的 CRC32 的低 4 个字节组合在一起的 sha1 再进行 base64
176	4	文件指纹 1，原始 APK 压缩包中第一个文件的 sha1 字符串的前 4 个字节
180	4	实际分析中，并没有用到
184	4	实际分析中，并没有用到
188	4	实际分析中，并没有用到
192	4	文件指纹 2，原始 APK 压缩包中第一个文件的 CRC32 的低 4 个字节转换成的字符串。
192+N*20	4	第 N 个文件，依次类推

整个签名校验的过程就是验证上面的4个文件指纹。在后面的分析中我们会真正的了解。

2.3.2 BB_HashTable 数据结构

BB_HashTable主要用于存储APK\\META-INF\ MANIFEST.MF文件内容的。APK\\META-INF\ MANIFEST.MF文件存储的是APK包中所有文件的SHA1签名：

```
Manifest-Version: 1.0

Name: AndroidManifest.xml
SHA1-Digest: KEbfcZikzEEld0mfzyxy4COF7AQ=

Name: assets/AZURE.png
SHA1-Digest: lcbV0PUldzpxDYiaXn5ePQDihB0=

Name: assets/App.zip
SHA1-Digest: Vl8E82WIZxngcR59wgfB029ARQU=

Name: assets/BLUE.png
SHA1-Digest: 48pxvOeYGxMU6vlDPSvopC8wKNs=

Name: assets/CYAN.png
SHA1-Digest: DOGgZDZe38eYauGGNAoeYd+tdrk=

Name: assets/GREEN.png
SHA1-Digest: E5Kx0EQK6ZzGdJU7s3yp2cmBG48=
```

图 2-3 [APK\\META-INF\ MANIFEST.MF]

其定义如下：

```
typedef struct BB_HashTable
{
    char* fileName;                    //文件名
    char* sha1;                       //对应文件的sha1签名
    int   noUsed0;
    int   noUsed1;
    int   noUsed2;
    UT_hash_table strhashTalbe;                //uthash结构

}BB_HashTable;
```

其中成员sha1用于文件指纹1的判断，同时用于组合文件指纹3。

2.3.3 ORG_file_Sign_list数据结构

这个结构主要是保存加固时APK的原始签名。如果实现防篡改功能，势必要保存原始APK的指纹信息，然后与当前APK的指纹信息进行对比，从而来判断是否被篡改。

这个结构对应的数据内容在如下目录：APK\assets\meta-data\ manifest.mf，其定义如下：

```
typedef struct ORG_file_Sign_list
{
    int index;                //链表索引   like 0 1 2 3...
    char* sha1First4Char;          //对应文件sha1字符串20个字符的0-3个字符(指纹3)
    char* sha1Second4Char;          //对应文件sha1字符串20个字符的4-7个字符
    char* sha1third4Char;          //对应文件sha1字符串20个字符的8-11个字符
    char* sha1fourth4Char;         //对应文件sha1字符串20个字符的12-15个字符
    char* sha1fifth4Char;          //对应文件sha1字符串20个字符的16-19个字符（指纹4）
    ORG_file_Sign_list* next;  //下一个节点 如果为0则表示是尾节点。

}ORG_file_Hash_list;

typedef struct ORG_fileSign
{
    //文件指纹1，原始APK的所有文件sha1的前4个字符组合在一起的sha1再进行base64
        char* allFileSha1First4Char;
    char* noUsed0;
    int   noUsed1;
    int   noUsed2;
    //文件指纹2，原始APK压缩包中第一个文件的CRC32的低4个字节转换成的字符串
        char* allFileCrc32Sha1;
    //单个文件对应的文件指纹信息，文件指纹3与文件指纹4
        ORG_file_Sign_list* orgFileSignListHead;
}ORG_fileSign;
```

本文件是识别指纹的核心文件，里面共存储了4种类型的文件指纹如下;

- 指纹1：原始APK单个文件sha1字符串的前4个字节
- 指纹2：原始APK单个文件对应的CRC32的低4个字节
- 指纹3：指纹1组合起来的SHA1的base64
- 指纹4：指纹2组合起来的SHA1的base64

2.2.4 CUR_file_crc32结构

本结构用于存储当前APK中每个文件的CRC32的数值，用于进行文件指纹2的比较。

```
typedef struct CUR_file_crc32
{
    char* fileName;                         //文件名
    int   crc32;                           //文件对应的CRC32
    CUR_file_crc32 next;                    //下一个节点
}CUR_file_crc32;
```


3 识别指纹的生成

从前面介绍中我们可以知道其存在4种类型的识别指纹，这些指纹存储在"APK\assets\meta-data\manifest.mf"中。同时还存在另外2个文件见下图：

图 3-1 [BB 加固之防篡改识别指纹文件]

其中ras.sig保存的是manifest.mf的RSA加密后的签名。Ras.pub保存的是RSA的公钥信息。BB不大可能拿到APP开发者签名的私钥，因此这个ras.pub应该是BB自己的公钥。这2个文件的作用其实就是验证原始的文件指纹是否被篡改。

3.1 识别指纹生成流程

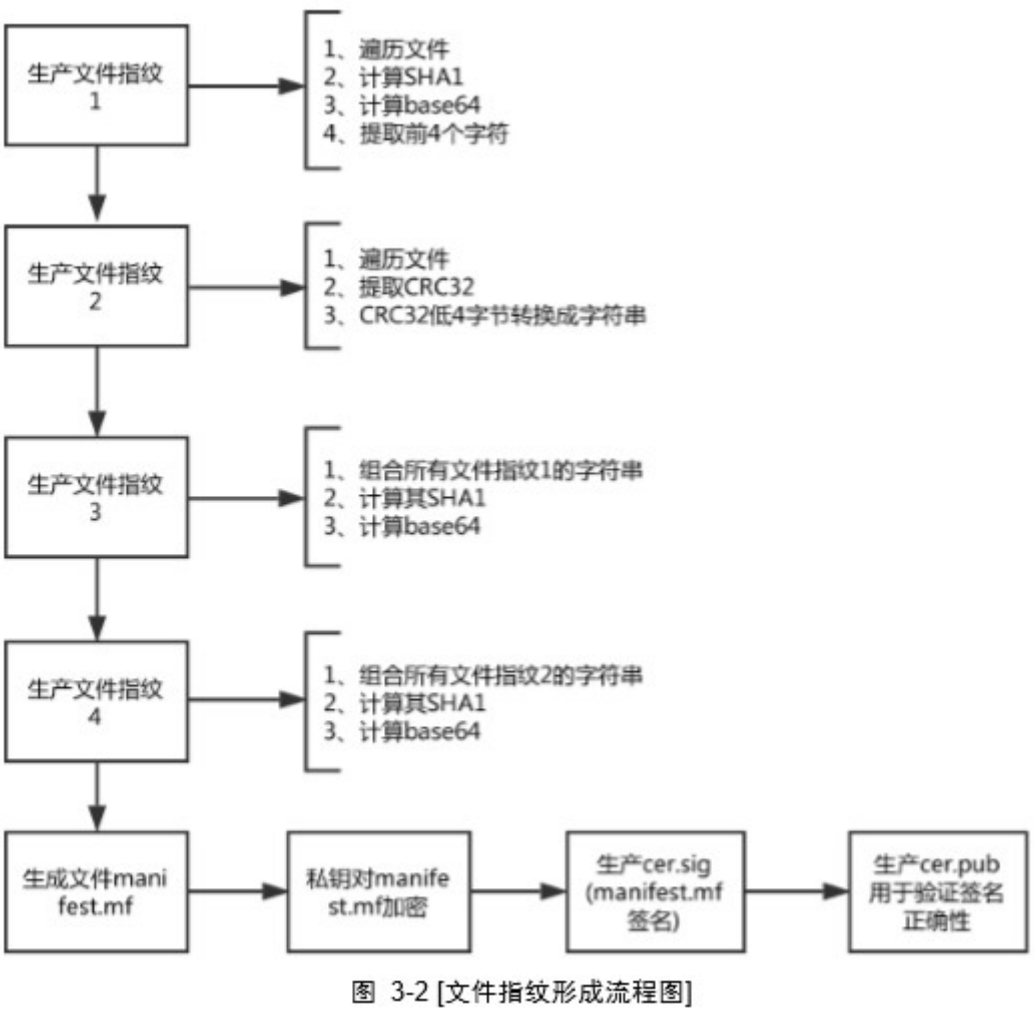


图 3-2 [文件指纹形成流程图]

4 验证流程

整个验证流程可以分为如下8个步骤：

1、 验证原始APK的文件指纹(manifest.mf)、文件指纹的签名(cert.sig)以及公钥文件(cer.pub)是否被篡改，如果被篡改，则终止应用；

2、 提取原始manifest.mf的4种类型的文件指纹；

3、 提取当前文件的sha1，其中包括当前文件指纹1；

4、 提取当前文件的CRC32，包括当前文件指纹2；

5、 对当前文件指纹1与原始文件指纹1比较，如果不一致，则终止应用；

6、 对当前文件指纹2与原始文件指纹2比较，如果不一致，则终止应用；

7、 将当前APK所有文件指纹1组合成一个字符串，并计算其整体的SHA1，然后再base64，即获得当前APK的文件指纹3，与原始的文件指纹3进行比较，如果不一致，则终止应用；

8、 将当前APK所有文件指纹2组合成一个字符串，并计算其整体的SHA1，然后再base64，即获得当前APK的文件指纹4，与原始的文件指纹4进行比较，如果不一致，则终止应用。

4.1 验证原始APK指纹文件的有效性

- manifest.mf：原始APK的文件指纹，文件格式见2.3.1；
- cert.sig：manifest.mf经过RSA加密后的签名；
- cer.pub：用于验证cert.sig的RSA公钥。

这里的RSA签名，并不是APK开发者的签名，而是BB自己的签名。

4.1.1 验证cert.pub文件正确性

在BB的libdexhelper.so文件中保存着cert.pub的MD5值，但是这个MD5值是经过加密后的，而且其加密的流程还是比较复杂的。解密流程首先 实现了一个从APK文件中提取文件的C语言版本解压程序，包括如下函数：

- dexZipOpenArchive //打开zip文件获得ZipArchive结构
- dexZipFindEntry //从ZIP中获取指定文件的入口结构
- dexZipGetEntryInfo //得到指定文件入口结构对应的信息
- ReadToBufFromZipFile //将指定文件的内容读到BUF中
- dexZipCloseArchive //关闭zip文件

利用上面的函数，将cert.pub读取到内存中，然后计算其MD5值。接着开始获取原始APK的MD5值，其经过如下4个步骤：

1、计算0x52600开始0x1000个字节的MD5 值；

2、构造一个斐波那契数列，1 1 2 3 5 8 13 21 34 55；

3、以斐波那契数列为索引从数组0x52600取出对应的数值，然后与步骤1得到的MD5值按字节亦或，获得一个0x10大小的KEY。实际上是下面RC4解密的KEY

4、利用步骤3获得的RC4 KEY ，对525E0开始的0x10大小的数据，进行RC4解密。解密的结果就是原始的cer.pub对应的MD5值。

然后，将2个MD5值进行比较，如果不一致，则终止应用。

4.1.2 验证manifest.mf正确性

经过如下步骤进行验证：

1、 读取manifest.mf文件到内存；

2、 读取cert.sig文件到内存；

3、 使用cert.pub对manifest.mf进行RSA签名；

4、 利用步骤3获得签名与cert.sig进行比较，如果不一致，则终止应用。

4 .2 里程碑

至此确定原始APK的识别指纹没有被篡改，下面开始进行真正的防篡改流程。

4.3 提取当前APK的文件指纹1

将APK\\META-INF\ MANIFEST.MF读到内存，并进行一行一行的遍历，将文件名和对应的sha1数值存储在结构BB_HashTable中。 BB_HashTable结构见2.3.2。

4.3 提取原始manifest.mf的4种类型的文件指纹

对原始manifest.mf文件进行解析，并将解析结果存储到ORG_file_Sign_list结构中，ORG_file_Sign_list结构见2.33。

4.4 提取当前文件的文件指纹2

遍历APK，过滤掉“assert\ meta-data”、“META-INF\”目录，并且获得每个文件的CRC32的数值，并存储在CUR_file_crc32结构中，CUR_file_crc32结构见2.3.4。

4.5 文件指纹验证

前面已经将原始APK的4种文件指纹，以及当前APK的4种文件指纹都获取到，然后就开始进行判断，只要有任何一个文件指纹不一致，则终止应用。

5 终止应用的方法

对于通用的终止应用的方法，一般可能采用kill、abort、exit等系统函数，BB显然知道这样去终止会带来很大的安全问题。因此其采用了另外一种终止方式。

当需要终止应用时，首先破坏堆栈内容，然后将PC指针指向存储APK路径名的位置，而其对应的代码是无法执行的，从而引发异常来使应用终止运行。

1、 提取原始文件的的文件指纹，并分为4种类型；

2、 使用RSA对原始文件指纹进行加密；

3、 同时在SO中保存RSA公钥的MD5值，而这个MD5值是经过多重加密存放的。里面用到了MD5\斐波那契数列\RC4\base64等加密算法；

4、 验证RSA公钥正确性；

5、 验证原始文件指纹的正确性，使用RSA公钥验证；

6、 至此确保原始文件的所有指纹都是正确的，下面开始进行真正的校验过程；

7、 获取原始文件的4种文件指纹

8、 获取当前文件的4种文件指纹

9、进行4种文件指纹的一一比较，只要有一个不符合，则终止进程。

参考资料：

1. http://blog.csdn.net/jiangwei0910410003/article/details/50402000

2. http://blog.csdn.net/devilcash/article/details/7230733

3. http://download.csdn.net/download/u_1_n_2_i_3/9722166

移动安全技术交流群（211730239）

上传的附件：

防篡改相关文件.zip（350.77kb，129次下载）

最新回复 (14)

流浪情人 2017-12-6 10:04 2 楼 0

大概能看懂，要是有工具就好了啊

极客

程IT龙 2017-12-6 10:06 3 楼 0

分析得真棒！ 有没有可以破解的思路?

极客

最新回复 (14)

oooAooo 9 2017-12-6 10:23 4 楼 0

学者

> 流浪情人 大概能看懂，要是有工具就好了啊

目前在防篡改上 普遍采用的是验证开发者公钥的信息，其中BB并没有采用这个方案，而是借鉴了android系统本身对签名的检验流程，此流程是在APK安装的过程中验证的，只不过BB在程序每次启动时，开一个 线程在线程中进行如上的验证。

由于是本地验证，为了保护指纹，采用了很多的加密措施，尤其对加密算法的秘钥做了进一步的处理。

采用这方案的，还有爱加密。其他家的加固基本上都是验证开发者签名的。包括360等。

对于BB的企业版加固，应该说是目前最强的，其DEX的VMP在百度和360的基础上，做了优化，不仅仅是smali操作码的混淆，对stringID\fieldID\typeID\protoID\meithodID也进行了混淆，而且将原始DEX进行了拆分，对于还原确实要困难的多，当然仍然也是可以被还原的。

oooAooo 9 2017-12-6 10:29 5 楼 0

学者

> oooAooo
>
> 目前在防篡改上 普遍采用的是验证开发者公钥的信息，其中BB并没有采用这个方案，而是借鉴了android系统本身对签名的检验流程，此流程是在APK安装的过程中验证的，只不过BB在程序每次启动 ...

逆向分析确实是个很枯燥的过程，我也是在学习的过程中，如果一定说思路的话，除了经验之外，需要对数据结构和算法掌握。在分析过程中如果能很块判断出数据结构和算法，逆向分析就会快得多。

PS :加密算法我很渣。需要像咱看雪算法大牛学习。

schroep 2 2017-12-6 10:50 6 楼 0

极客

分析的是最新版本呢嘛？大大

tDasm 7 2017-12-6 14:21 7 楼 0

极客

BB加固对加密认识好像有误区？以为算法越多、算法越强那么加固就越强？
我只知道内存dump是不需要关心算法的（除非在内存中也一直是加密状态）。既然dex在内存中可以拆分那么也就可以在内存中组装。谁会去直接修改你的加密文件呢？所以那么多相互效验毫无意义。别人都是dump后该干吗就干吗！
个人认为BB难点应该是VMP解释smali，其他应该是画蛇添足。

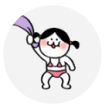
Zkeleven 3 2017-12-7 12:47 8 楼 0

极客

> tDasm BB加固对加密认识好像有误区？以为算法越多、算法越强那么加固就越强？我只知道内存dump是不需要关心算法的（除非在内存中也一直是加密状态）。既然dex在内存中可以拆分那么也就可以在内存中组装。谁会去直 ...

楼主对防篡改有误区，防篡改不是源代码保护，而是防止apk被重打包，不是用来保护dex的，很多时候不需要管dex，只需要改apk里面一些xml或者资源文件就行。

最新回复 (14)



Caln 2 2017-12-7 14:16 9 楼 0

大牛

> Zkeleven 楼主对防篡改有误区，防篡改不是源代码保护，而是防止apk被重打包，不是用来保护dex的，很多时候不需要管dex，只需要改apk里面一些xml或者资源文件就行。

????

tDasm 7 2017-12-7 14:34 10 楼 0

极客

> Zkeleven
>
> 楼主对防篡改有误区，防篡改不是源代码保护，而是防止apk被重打包，不是用来保护dex的，很多时候不需要管dex，只需要改apk里面一些xml或者资源文件就行。

真不知道谁对防篡改有误区？！！
打个比方：你是靠衣服遮丑，如果你的衣服脱光了，你还靠什么遮丑？

Zkeleven 3 2017-12-7 15:42 11 楼 0

极客

> tDasm Zkeleven 楼主对防篡改有误区，防篡改不是源代码保护，而是防止apk被重打包，不是用来保护dex的，很多时候不需要管dex，只需要改apk里面一些xml ...

没明白你的比方，还有你这么激动干嘛。。

无边 5 2017-12-7 17:15 12 楼 0

极客

> Zkeleven 楼主对防篡改有误区，防篡改不是源代码保护，而是防止apk被重打包，不是用来保护dex的，很多时候不需要管dex，只需要改apk里面一些xml或者资源文件就行。

开始我也没看懂，看了官网介绍后才知道楼主说的是官网说的防篡改保护里的防二次打包，其实只要说防止apk二次打包就明确了，防篡改保护范围太大，容易误导。还有楼主说的就是签名认证的原理过程，我估计过签名认证的会不会这些都没什么影响。

xiling 2 2018-5-23 11:27 13 楼 0

极客

对..哎..发现现在越来越多的事情总是绕着圈说一堆事....
这个其实就是防二次打包的....

GuardianCode 2 2018-5-24 14:24 14 楼 0

极客

六种加密算法？可能对什么是加密 什么是编码 什么是哈希 概念都很模糊

myeanngg 2 2018-5-25 21:45 15 楼 0

极客

就是考验别人对加密算法的熟悉程度

游客

登录 | 注册 方可回帖

回帖 表情 高级回复

返回

27