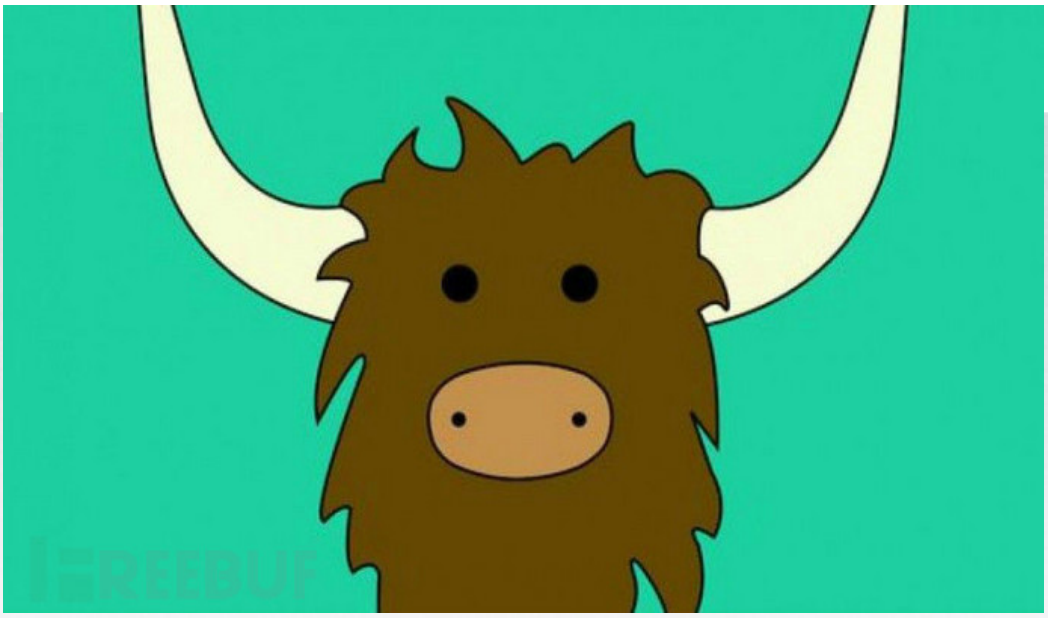
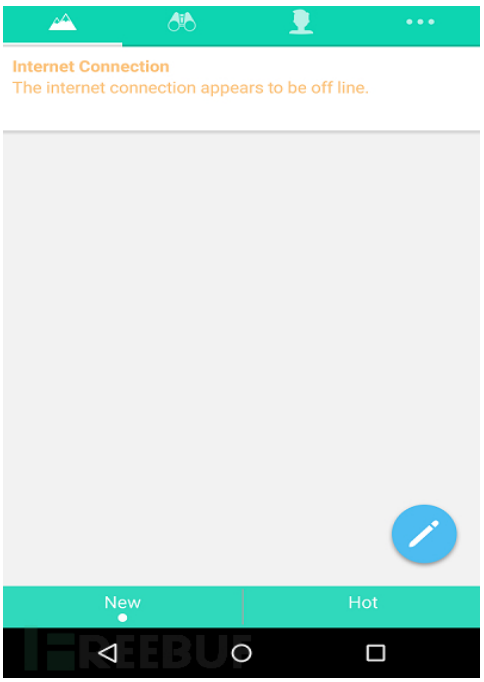


**本文主要对安卓平台上的匿名社交媒体应用Yik Yak进行逆向分析，在分析的过程中发现该APP用到了代码混淆、字符串混淆、防签名篡改校验等技术，最后通过整个分析过程总结了APP分析的流程和方法。**

## 0×01 Yik Yak介绍

每隔一段时间，我就会遇到一个实现了一些加固技术的APP，从而使逆向更加有趣。最近，当我尝试为Yik Yak代理API 请求时，就遇到了这种情况。其中，Yik Yak是一款流行的专用于移动平台的社交媒体应用，它允许半径为5英里（通常是大学校园）之内的半匿名用户之间进行交流。

在执行self-MITM攻击时打开应用程序，能够有效地杀掉所有API通信，通常来说是SSL pinning的一个指标。




## 0×02 混淆措施

在反编译该APK文件且检查Java源代码后，很明显地发现该APP的开发人员还使用了一种混淆工具来优化/保护他们的构建。这里有一个示例方法：

```
public int hashCode() {
            if (e == null) {
                i1 = 0;
            } else {
                i1 = e.hashCode();
            }
            l2 = f;
            i3 = g;
            if (h) {
                c1 = '\u04CF';
            } else {
                c1 = '\u04D5';
            }
            j3 = fs.a(i);
            if (j == null) {
                j1 = 0;
            } else {
                j1 = j.hashCode();
            }
            k3 = Arrays.hashCode(k);
```

```
                i3  =  Arrays.hashCode(l);
                i4  =  Arrays.hashCode(m);
                if  (n  ==  null)  {
                        k1  =  0;
                }  else  {
                        k1  =  n.hashCode();
                }
                if  (o  ==  null)  {
                        l1  =  0;
                }  else  {
                        l1  =  o.hashCode();
                }
                j4  =  (int)  (p  ^  p  >>>  32);
                if  (q  !=  null)  {
                        i2  =  q.hashCode();
                }
                return  ((((((l1  +  (k1  +  ((((j1  +  ((c1  +  (((i1  +  ((j2  +  52
7)  *  31  +  k2)  *  31)  *  31  +  12)  *  31  +  i3)  *  31)  *  31  +  j3)  *  3
1)  *  31  +  k3)  *  31  +  13)  *  31  +  i4)  *  31)  *  31)  *  31  +  j4)  *  3
1  +  i2)  *  31  +  Arrays.hashCode(r))  *  31  +  s)  *  31  +  fs.a(t))  *  3
1  +  b();
        }
```

注意，变量、类和方法名已经从原始的友好的形式进行了重命名。随着我查看更多的代码，发现它也似乎还进行了字符串常量混淆操作，这种特性存在于DexGuard等第三方工具中。通常来说，虽然代码混淆是一个很好的习惯，但它只能通过使代码变得难以阅读，从而能够放缓攻击者的逆向进度。根据我的经验，混淆也能使Java反编译结果变得不可靠，所以我大多时候会在smali层面进行分析。

## 0×03 搜索字符串

接下来，我开始对smali源文件执行grep操作，以搜索与常见SSL pinning实现相关的字符串。很快，我找到了自认为是pinning校验的代码：

```
.method  public  checkServerTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/Strin
g;)V
        .locals  3

        .prologue
        const/4  v2,  0x0
```

```
        .line  153
        iget-object  v0,  p0,  LCG;->e:Ljava/util/Set;

        aget-object  v1,  p1,  v2

        invoke-interface  {v0,  v1},  Ljava/util/Set;->contains(Ljava/lang/Object;)Z

        move-result  v0

        if-eqz  v0,  :cond_0

        .line  163
        :goto_0
        return-void

        .line  160
        :cond_0
        invoke-direct  {p0,  p1,  p2},  LCG;->a([Ljava/security/cert/X509Certificate;Ljav
a/lang/String;)V

        .line  161
        invoke-direct  {p0,  p1},  LCG;->a([Ljava/security/cert/X509Certificate;)V

        .line  162
        iget-object  v0,  p0,  LCG;->e:Ljava/util/Set;

        aget-object  v1,  p1,  v2

        invoke-interface  {v0,  v1},  Ljava/util/Set;->add(Ljava/lang/Object;)Z


        goto  :goto_0
.end  method
```

通过编辑上面的方法直接返回void，我绕过了此方法。但在构建、签名，并安装了这个新的APK后，我得到了与上面相同的“网络连接（Internet Connection）”错误。在以不同的编辑/构建方式尝试多次，且每次都得到相同的结构之后，我开始怀疑Yik Yak使用了一些篡改检测逻辑（一种包签名验证），以此来防止自己被他人逆向分析。于是，我安装了一个未修改但经过重签名的APK，仍然出现相

同的错误，这一结果使我确认了我怀疑的内容。

## 0x04 搜索决策点

为了绕过篡改检测，我改变了焦点并开始搜索它的决策点。在安卓中，开发人员可以通过以下方式使用PackageManager类访问包签名：

```
PackageManager pm = context.getPackageManager();
String packageName = context.getPackageName();

Signature[] sigs = pm.getPackageInfo(packageName, PackageManager.GET_SIGNATURES).signatures;
```

因为签名是作为android.content.pm.Signature的实例返回的，所以我搜索了代码并发现下面的方法：

```
public static Signature[] a(Context context)
        {
                if (context != null) goto _L2; else goto _L1
_L1:
                PackageManager packagemanager;
                return null;
_L2:
                if ((packagemanager = context.getPackageManager()) == null) goto _L1; else goto _L3
_L3:
                try
                {
                        context = packagemanager.getPackageInfo(context.getPackageName(), 64);
                }
                // Misplaced declaration of an exception variable
                catch (Context context)
                {
                        context.printStackTrace();
                        return null;
                }
                if (context == null) goto _L1; else goto _L4
_L4:
```

```
            context = ((PackageInfo) (context)).signatures;
            return context;
        }
```

很显然，这个方法用作一个包装器来获取当前构建的签名。注意，在第12行，值64正确匹配了PackageInfo.GET_SIGNATURES的常量值。搜索这个类的使用后，得到了下面的几个结果：

```
user@rw:/var/www/yikyak/smali$ grep -r "Ab;"
sb.smali:    invoke-static {v1}, LAb;->a(Landroid/content/Context;)[Landroid/content/pm/Signature;
Ab.smali:.class public LAb;
sk.smali:    invoke-static {v1}, LAb;->a(Landroid/content/Context;)[Landroid/content/pm/Signature;
sk.smali:    invoke-static {v3}, LAb;->b(Landroid/content/Context;)Z
sk.smali:    invoke-static {v3}, LAb;->a(Landroid/content/Context;)[Landroid/content/pm/Signature;
user@rw:/var/www/yikyak/smali$
```

## 0×05 提取签名

我没有试图解决多决策点，而是决定通过修改上述方法返回官方Yik Yak构建的签名，以此来欺骗包签名。为了做到这一点，我需要知道应用程序正在寻找的签名。在上述结果中，我简要地搜索了一下那些代码，但是并没有发现硬编码签名（可能是由于混淆）。我没有进一步搜索（或调试）代码，而是跑了一个脚本，该脚本会从一个给定的APK中提取签名：

```
user@rw:/var/www/yikyak/tools/GetAndroidSig$ javac *.java
user@rw:/var/www/yikyak/tools/GetAndroidSig$ jar -cvfe GetAndroidSig.jar Main .
added manifest
adding: Main.class(in = 4673) (out= 2596)(deflated 44%)
adding: Main.java(in = 4500) (out= 1488)(deflated 66%)
adding: yikyak_orig.apk(in = 21105312) (out= 19776095)(deflated 6%)
user@rw:/var/www/yikyak/tools/GetAndroidSig$ java -jar GetAndroidSig.jar yikyak_orig.apk
yikyak_orig.apk
Cert#: 0  Type:X.509
Public key: Sun RSA public key, 1024 bits
  modulus: 101032039018650828168239531910738579735352902090943931726535911635326971340454332667345617568386184986
626762279934590712012253855985757182431476954833177465427320103924373498650233394538202016604613886359172999826854
545516708599816618372987203616660912829946673744520520057889142517681295001929282944121555457527
  public exponent: 65537
Hash code: 1465535706 / 0x575a4cda
To char: 3082019d30820106a003020102020452ab687f300d06092a864886f70d0101050500301231103000e06035504031307776696c6c6
9616d3020170d313331323133323030303531395a180f323131333131313932303035313935a30123110300e06035504031307776696c6c69616
d30819f300d06092a864886f70d01010105000381d0030818902818100fdfd8a1c6319b8d45445dc9c28a89600062dd00ad14c5ee3fac8
d4812d5dfa3a5c6e534f242d5e91d6acb1807d618d44731973c4f69c328b6b755962810ed2cf8ff19fa5c6de40a34be5e92c6686e772fa86
4784e74144465272c260f877395df37b897e8147bbcdce15b8f11ee125c82bf9d2de9beb92056edea6f301d15f70203010001300d06092a8
54886f70d0101050500038181000a2f44f1a8d78b4d1965f0e60f9ef10826827ae131e6c4a3f976fc85f36f94578a698f904fd0a37a690f3
dd338c16c3e408d77670543bb5b022d7c1bc86a0574e3e593092f1e06de141f04f6a68d78dbc5aa36f0a82062ecb03c1e7285a55b5ccfea5
3c193572d8d7542ca7a31748aabc7edff7990048a11ae5ef090074c9b25
user@rw:/var/www/yikyak/tools/GetAndroidSig$
```

## 0×06 绕过签名校验

在得到期待的签名后，我修改了上述smali方法，让它直接返回该签名：

```
.method public static a(Landroid/content/Context;)[Landroid/content/pm/Signature;
        .locals 4

        .prologue
        const/4 v0, 0x0
```

```
        const-string  v0,  "3082019d30820106a003020102020452ab687f300d06092a864886f70d010
10505003012311030 0e060355040313077769 6c6c69616d3020170d313331323133323030353139 5a180f3231
313331313139323030353139 5a30123110300e060355040313077769 6c6c69616d30819f300d06092a864886f
70d01010105000381 8d0030818902818100 fdfd8a1c6319b8d45445dc9c28a89600062dd00ad14c5ee3fac8d
4812d5dfa3a5c6e534f242d5e91d6acb1807d618d44731973c4f69c328b6b755962810ed2cf8ff19fa5c6de40
a34be5e92c6686e772fa864784e74144465272c260f877395df37b897e8147bbcdce15b8f11ee125c82bf9d2d
e9beb92056edea6f301d15f70203010001300d06092a864886f70d0101050500038181000a2f44f1a8d78b4d1
965f0e60f9ef10826827ae131e6c4a3f976fc85f36f94578a698f904fd0a37a690f3dd338c16c3e408d776705
43bb5b022d7c1bc86a0574e3e593092f1e06de141f04f6a68d78dbc5aa36f0a82062ecb03c1e7285a55b5ccfe
a58c193572d8d7542ca7a31748aabc7edff7990048a11ae5ef090074c9b25"

        .line  11
        .local  v0,  "fake":Ljava/lang/String;
        const/4  v2,  0x1

        new-array  v1,  v2,  [Landroid/content/pm/Signature;

        const/4  v2,  0x0

        new-instance  v3,  Landroid/content/pm/Signature;

        invoke-direct  {v3,  v0},  Landroid/content/pm/Signature;-><init>(Ljava/lang/Stri
ng;)V

        aput-object  v3,  v1,  v2


        .line  13
        .local  v1,  "sig":[Landroid/content/pm/Signature;
        return-object  v1
.end  method
```

这是Java中相同的代码：

```
public  static  Signature[]  a(Context  context)
{
        String  fake  =  "3082019d30820106a003020102020452ab687f300d06092a864886f70d01010
505003012311030 0e060355040313077769 6c6c69616d3020170d313331323133323030353139 5a180f323131
333131313932303035313935 5a30123110300e060355040313077769 6c6c69616d30819f300d06092a864886f70
```

```
d010101050003818d00308189028181008fdfd8a1c6319b8d45445dc9c28a89600062dd00ad14c5ee3fac8d48
12d5dfa3a5c6e534f242d5e91d6acb1807d618d44731973c4f69c328b6b755962810ed2cf8ff19fa5c6de40a3
4be5e92c6686e772fa864784e74144465272c260f877395df37b897e8147bbcdce15b8f11ee125c82bf9d2de9
beb92056edea6f301d15f70203010001300d06092a864886f70d010105050003818100a2f44f1a8d78b4d196
5f0e60f9ef10826827ae131e6c4a3f976fc85f36f94578a698f904fd0a37a690f3dd338c16c3e408d77670543
bb5b022d7c1bc86a0574e3e593092f1e06de141f04f6a68d78dbc5aa36f0a82062ecb03c1e7285a55b5ccfea5
8c193572d8d7542ca7a31748aabc7edff7990048a11ae5ef090074c9b25";

        Signature[]  sig  =  new  Signature[]{new  Signature(fake)};

        return  sig;
}
```

现在，我已经绕过了包签名检查，于是安装新的构建以进行测试。不幸的是，出现了同样的错误。因为这可能是因为我错过了一些额外的pinning代码，所以我再一次搜索了整个源码，最终发现这个方法：

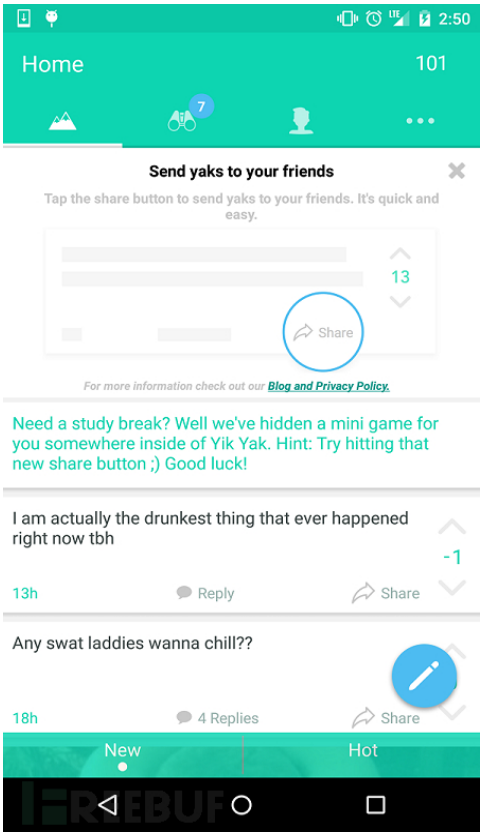
```
public  void  a(String  s,  List  list)
        {
                List  list1;
                boolean  flag;
                flag  =  false;
                list1  =  (List)b.get(s);
                if  (list1  !=  null)  goto  _L2;  else  goto  _L1
_L1:

                return;
_L2:
                int  l  =  list.size();
                int  i  =  0;
label0:
                do
                {
label1:
                        {
                                if  (i  >=  l)
                                {
                                        break  label1;
                                }
                                if  (list1.contains(a((X509Certificate)list.get(i))))
```

```
                                if  (list1.contains(a((X509Certificate)list.get(i))))
                                {
                                        break  label0;
                                }
                                i++;
                        }
                }  while  (true);
                if  (true)  goto  _L1;  else  goto  _L3
_L3:
                StringBuilder  stringbuilder  =  (new  StringBuilder()).append("Certificate  pinning  failure!").append("\n    Peer  certificate  chain:");
                int  i1  =  list.size();
                for  (int  j  =  0;  j  <  i1;  j++)
                {
                        X509Certificate  x509certificate  =  (X509Certificate)list.get(j);
                        stringbuilder.append("\n        ").append(a(((Certificate)  (x509certificate)))).append(":  ").append(x509certificate.getSubjectDN().getName());
                }

                stringbuilder.append("\n    Pinned  certificates  for  ").append(s).append(":");
                i1  =  list1.size();

                for  (int  k  =  ((flag)  ?  1  :  0);  k  <  i1;  k++)
                {
                        s  =  (Dx)list1.get(k);
                        stringbuilder.append("\n        sha1/").append(s.b());
                }

                throw  new  SSLPeerUnverifiedException(stringbuilder.toString());
        }
```

通过“return-void”绕过这个方法使得pinning实现被禁用，从而我能够成功地代理APP的API请求。

```
GET  https://notify.yikyakapi.net/api/getAllForUser/***REMOVED***  HTTP/1.1
Host:  notify.yikyakapi.net
Connection:  Keep-Alive
Accept-Encoding:  gzip
```

```
Cookie:  __cfduid=***REMOVED***
User-Agent:  Dalvik/2.1.0  (Linux;  U;  Android  5.1.1;  Nexus  6  Build/LMY48M)  3.0
```




## 0×07 总结

在信息安全的世界中，充满了公司糟糕地处理软件安全的例子，但是可以明显地看到，像Yik Yak这样的公司还是一直在努力提高软件的安全性的，至少在安卓APP方面是这样。

***参考来源：randywestergren，FB小编JackFree编译，转载请注明来自FreeBuf黑客与极客（FreeBuf.com）**