ADAM: An Automatic & Extensible Platform To Stress Test Android Anti-Virus Systems

John C.S. Lui
Spark ZHENG Min
Patrick P.C. Lee
Android Malware
Up 3,325% in 2011

1. This past year, we saw a significant increase in mobile malware.

   A. Juniper Networks Mobile Threat Center: 400 to 13,302.
   B. Antiy: 12,000.
   C. Tencent: 10,000.

2. Spyware and premium rate SMS Trojans are the most popular Android malware.
Motivation Of ADAM System

Both academic community and commercial anti-virus companies proposed many methodologies and products. How to assess the effectiveness of these defense mechanisms? Especially malware mutation.
Related Work Of ADAM System

1. No byte code level obfuscation research on Android malware before.
   e.g. Jha, Testing Malware Detectors, 04; Moser, Limits of Static Analysis for Malware Detection. ACSAC 07

2. No large-scale evaluation on 40+ anti-virus and 200*8+ malware on Android before.
   e.g. Felt. A survey of mobile malware in the wild, SPSM 11

3. No automatic Anti-virus test system on Android before.
   e.g. Jiang, Dissecting Android Malware, Oakland12.
We propose: **ADAM** which can *automatically* transform an original sample to different *variants* via repackaging and obfuscation techniques. Then *stress test* anti-virus products.
Introduction
To Apk and Disassemble

1. The .apk file contains all of the information necessary to run the application on a device or emulator, such as compiled .dex file, a binary version of the AndroidManifest.xml file, compiled resources (resources.arsc) and uncompiled resource files for your application.

2. The disassemble process takes the Dalvik opcodes of a .dex file and converts them into low-level and human readable instructions. Typically, the decoded .smali files can be rebuilt again back to a .dex file.
System Design Of ADAM

   For original sample and variants.

2. Automated transformation.
   no source code need.

3. Extensibility.
   Plug-in new detection systems or obfuscation techniques.
Obfuscation Technique Of Repackaging

Repackaging methods that work directly on an input .apkg file and regenerate a different .apkg file without modifying the source code of the input .apkg file.

<1>. Alignment.
The process only changes the cryptographic hash of the .apkg file.

<2>. Re-sign.
An .apkg file can be re-signed multiple times with different certificates.

<3>. Rebuild.
Disassembles an .apkg file and rebuilds the assembly code (without being modified) into another .apkg file.
Evaluation Of Repackaging

1. We collect a total of 222 distinct Android malware samples.

2. Online Engine: VirusTotal.

3. Note that VirusTotal hosts over 40 anti-virus products, and our study only focuses on the top 10 products.
### Analysis Of Rebuild Technique

<table>
<thead>
<tr>
<th>Class 0:</th>
<th>public final mars.testbc.R$attr</th>
<th>public final mars.testbc.R$attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1:</td>
<td>public final mars.testbc.R$drawable</td>
<td>public mars.testbc.TestActivity</td>
</tr>
<tr>
<td>Class 2:</td>
<td>public final mars.testbc.R$id</td>
<td>public final mars.testbc.R$id</td>
</tr>
<tr>
<td>Class 3:</td>
<td>public final mars.testbc.R$layout</td>
<td>public final mars.testbc.R$drawable</td>
</tr>
<tr>
<td>Class 4:</td>
<td>public final mars.testbc.R$string</td>
<td>public mars.testbc.TestReceiver</td>
</tr>
<tr>
<td>Class 5:</td>
<td>public final mars.testbc.R</td>
<td>public final mars.testbc.R$layout</td>
</tr>
<tr>
<td>Class 6:</td>
<td>mars.testbc.TestActivity$Broadcast</td>
<td>mars.testbc.TestActivity$Broadcast</td>
</tr>
<tr>
<td>Class 7:</td>
<td>public mars.testbc.TestActivity</td>
<td>public final mars.testbc.R$string</td>
</tr>
<tr>
<td>Class 8:</td>
<td>public mars.testbc.TestReceiver</td>
<td>public final mars.testbc.R</td>
</tr>
</tbody>
</table>

#### Original vs. Rebuild

<table>
<thead>
<tr>
<th>Original</th>
<th>Rebuild</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 0:</td>
<td>public final mars.testbc.R$attr</td>
</tr>
<tr>
<td>Class 1:</td>
<td>public final mars.testbc.R$drawable</td>
</tr>
<tr>
<td>Class 2:</td>
<td>public final mars.testbc.R$id</td>
</tr>
<tr>
<td>Class 3:</td>
<td>public final mars.testbc.R$layout</td>
</tr>
<tr>
<td>Class 4:</td>
<td>public final mars.testbc.R$string</td>
</tr>
<tr>
<td>Class 5:</td>
<td>public final mars.testbc.R</td>
</tr>
<tr>
<td>Class 6:</td>
<td>mars.testbc.TestActivity$Broadcast</td>
</tr>
<tr>
<td>Class 7:</td>
<td>public mars.testbc.TestActivity</td>
</tr>
<tr>
<td>Class 8:</td>
<td>public mars.testbc.TestReceiver</td>
</tr>
</tbody>
</table>

**CUHK CSE**
Code obfuscation changes the size and content of the .apk file by rebuilding the assemble code, but without modifying the logical behavior.

1. Inserting defunct methods. The rationale of this obfuscation technique is to modify the method table in the Dalvik bytecode.

2. Renaming methods. We obfuscate the method name with a different string, and hence change the signature that is generated by the method name.
Obfuscation Technique Of Code Obfuscation

<3>. Changing control flow graphs. We modify the CFG without changing the logic behavior of a .smali file and so as to change its CFG signature.

<4>. Encrypting constant strings. We encrypt all constant strings that we find in a .smali file, and decrypt them when they are being processed by modifying the invoking instructions.
We can encrypt a string “DecryptString” in a TextView control by subtracting all bytes by 10. The encrypted string will become “:[Yhofjljh_d]”.

We then add the decryption method decrypt (i.e., by adding all bytes by 10) before the TextView control is called.

```
#direct methods
.method public static DecryptString
    (Ljava/lang/String;)Ljava/lang/String;
    ...
    const-string v1, ":[Yhofjljh_d]"
    ...
    invoke-static { v1},
    Lcom/test;->DecryptString
    (Ljava/lang/String;)Ljava/lang/String;
    move-result-object v1
    invoke-virtual {v0, v1}, Landroid/
    widget/TextView;->setText
    (Ljava/lang/CharSequence;)V
```

Fig. 5. Encrypting a constant string.
## Evaluation Of Repackaging and Code Obfuscation

**AV Products** | **Original** | **Alignment** | **Re-sign** | **Rebuild**
--- | --- | --- | --- | ---
Kaspersky | 95.95% | 94.34% | 94.59% | 94.76%
F-Secure | 95.50% | 95.75% | 95.05% | 91.90%
Emsisoft | 94.59% | 93.87% | 93.69% | 75.24%
Ikarus | 94.59% | 94.34% | 93.69% | 75.24%
GData | 94.14% | 93.87% | 93.69% | 90.95%
TrendMicro | 94.14% | 91.98% | 92.79% | 77.62%
NOD32 | 92.79% | 88.68% | 88.29% | 95.24%
Sophos | 92.79% | 94.81% | 94.14% | 78.10%
Antiy-AVL | 92.34% | 91.98% | 89.19% | 72.38%
Fortinet | 90.99% | 89.15% | 88.74% | 71.43%
**Overall Average** | **93.78%** | **92.88%** | **92.39%** | **82.29%**

**AV Products** | **Insert** | **Rename** | **Change CFG** | **Str. Encrypt**
--- | --- | --- | --- | ---
Kaspersky | 93.81% | 73.33% | 94.76% | 90.95%
F-Secure | 90.00% | 90.00% | 90.48% | 68.57%
Emsisoft | 83.81% | 26.67% | 82.86% | 25.24%
Ikarus | 83.81% | 26.67% | 83.33% | 25.24%
GData | 90.95% | 90.48% | 91.43% | 88.10%
TrendMicro | 61.90% | 61.90% | 63.81% | 35.71%
NOD32 | 95.24% | 91.90% | 95.24% | 90.48%
Sophos | 54.29% | 54.29% | 54.76% | 49.05%
Antiy-AVL | 70.00% | 19.05% | 67.14% | 19.52%
Fortinet | 48.57% | 15.71% | 42.86% | 16.67%
**Overall Average** | **77.24%** | **55.00%** | **76.67%** | **50.95%**

Repackaging
November 2011

code obfuscation

---

**SMARTPHONE Security**

---

Spark
ZHENG Min
Discussion Of ADAM System

1. Signature coverage.
   We cannot verify if all anti-virus systems that we tested on VirusTotal apply the same detection logic as in their mobile versions.

2. Distribution model.
   It is generally difficult to distribute malicious applications through the official AndroidMarket. However, we believe that hackers can upload any malware to the third-party markets.
Future Work Of ADAM System

1. We try to extend our system to support mobile version anti-virus products and dynamic analysis system.

2. We try to add a new function that explore the logic of anti-virus engine.