Things You May Not Know About Android
(Un)Packers:
A Systematic Study based on Whole-System Emulation

Yue Duan, Mu Zhang,
Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li,
Xueqiang Wang, XiaoFeng Wang

University of California, Riverside     Cornell University
Grammatech. Inc     Indiana University Bloomington    Peking University
Motivation

• What is Android packing?

Android packing is a technique that helps apps hide their original code from being analyzed.

2.4 MB to 1 KB
Motivation

• Why important to security community?
• Packing techniques can indeed help malware sneak into Google Play[1][2].

Report from Checkpoint[2]

Most malware found on Google Play contains only a dropper that later downloads the real malicious components to the device. Charger, however, uses a heavy packing approach which it harder for the malware to stay hidden, so it must compensate with other means. The developers of Charger gave it everything they had to boost its evasion capabilities and so it could stay hidden on Google Play for as long as possible.
Motivation

- We performed the first large-scale measurement study to better understand Android packing.
  - 7 popular commercial packers including Ali, apkprotect, baidu, Bangcle, ijiami, Qihoo and Tencent
  - 5 recent malicious apps
  - 93,910 Android malware from VirusTotal
  - 5 representative apps, consider them as ground truth and perform diff analysis
  - 3 state-of-the-art Android unpackers
What do we want to study?

• Question set 1: High Level Landscape
• Question set 2: Detailed Behavioral Analysis
• Question set 3: Evolution of Android packing
• Question set 4: Existing defeating techniques
Challenges

• **NO** existing tool can be directly leveraged to conduct this study.

• We need a tool that could provide
  
  • reliable and generic unpacking unknown packers
  • correctly handle native, Java as well as JNI
  • be able to understand behaviors
Challenges

• Based on our study, state-of-the-art Android unpackers have fundamental design limitations.
  • Signature-based: Kisskiss[9]
  • Hooking-based: DexHunter[7]
  • Dalvik data structure dumping: AppSpear[8]

• Limitations
  • signatures are not reliable
  • cannot handle multi-layer unpacking
  • cannot support ART
DroidUnpack System

- Key idea
  - Generic unpacking based on memory operation monitoring
  - Reconstruct Java level execution
  - VM based approach

![Diagram](image-url)
DroidUnpack System

• Compared to Renovo[5]
  • reconstruct Java level info
• Compared to Droidscope[6]
  • retrieve ART view
DroidUnpack System

- Reconstructing ART Semantic View
  - Compiled Java functions
  - Interpreted Java functions

Invoke()

DoCall()

ArtMethod object → ArtMethod's declaring class → DexCache → DexFile object

Correct timing

Reliable data structures within runtime

Module
Function info
Code
Etc.
Findings - High level landscape

Fig. 2: Yearly distribution.

Fig. 3: Packer distribution.

Fig. 4: Trend of packer distribution.
Findings - Detailed analysis

- Detailed analysis: Commercial packers have adopted many unique yet unreported features for anti-unpacking.

<table>
<thead>
<tr>
<th>TABLE I: Commercial packer behavior.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context switching via JNI</td>
</tr>
<tr>
<td>Native/DEX obfuscation</td>
</tr>
<tr>
<td>Pre-compilation</td>
</tr>
<tr>
<td>Multi-layer unpacking</td>
</tr>
<tr>
<td>libc.so hooking</td>
</tr>
<tr>
<td>Self modification</td>
</tr>
<tr>
<td>Component hijacking vulnerability</td>
</tr>
<tr>
<td>Information leakage</td>
</tr>
</tbody>
</table>
Findings - Detailed analysis

- multi-layer unpacking
  - not necessarily a one-time effort

TABLE II: Multi-layer unpacking.

<table>
<thead>
<tr>
<th></th>
<th># of layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>apkprotect</td>
<td>1</td>
</tr>
<tr>
<td>Ali</td>
<td>1</td>
</tr>
<tr>
<td>Bangcle</td>
<td>9</td>
</tr>
<tr>
<td>ijiami</td>
<td>4</td>
</tr>
<tr>
<td>Qihoo</td>
<td>4</td>
</tr>
<tr>
<td>Tencent</td>
<td>40</td>
</tr>
</tbody>
</table>
Findings - Detailed analysis

- libc hooking (Bangcle)
  - a way of defeating unpackers.
  - packers are evolving to defeat unpackers.

memory operation monitoring using VMI
Findings - Detailed analysis

• Commercial packers have led to severe security vulnerability and data breach.
  • Upon packing, commercial packers change the behaviors of the program
  • Key idea: Is the change secure?
Findings - Detailed analysis

- Component hijacking vulnerability (Qihoo)
  - one vulnerable component is packed and added by the packer into the app
  - analyze the hidden component extracted by DroidUnpack
  - turn a perfectly secure app into a vulnerable app
  - acknowledged by Qihoo and awarded ~$8000
Findings - Detailed analysis

- Component hijacking vulnerability
  - can arbitrarily replace any file within a packed app from a designated server

```java
Intent intent = new Intent();

intent.setClassName("com.example.hellojni", "com.qihoo.util.CommonService");
intent.setAction("com.qihoo.commonservice.SERVICE_download");

Bundle bundle = new Bundle();
bundle.putString("md5", "E695392B43690F52752AD0D675E73427");
bundle.putString("url", "our server");
bundle.putString("name", "libjiagu.so"); // executable
bundle.putString("path", "/data/data/com.example.hellojni/jiagu/";
bundle.putLong("contentLength", 40544412);
bundle.putBoolean("init_only", false);
intent.putExtra("download", bundle);

startService(intent);
```
Findings - Detailed analysis

• Information leakage* (Tencent)
  • upon packing, it adds six new permissions to the original apps
  • *collect* sensitive user data and *send* them back via an insecure HTTP connection
  • utilize DroidUnpack to dump the hidden code
  • used FlowDroid to analyze

*This issue was identified by static analysis. We tried to contact Tencent to confirm but no reply so far.
Findings - Detailed analysis

- Impact of the security issues
  - Component Hijacking Vulnerability
    - Gaode Navi is actively used by more than 500 million users as their daily navigation app.
    - Qianniuniu finance has been downloaded for more than 3 million times
  - Information Leakage
    - QQ has more than 800 million active users
Findings - Detailed analysis

• Since everyone can use commercial packers, can they be exploited?

Protection technique claimed! However..

<table>
<thead>
<tr>
<th>TABLE III: Security scrutiny.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware defense failure</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

1 apkprotect is not on-line service and has no prevention for malware or plagiarism.
2 Qihoo detected first attempt and blocked further malware submission.

<table>
<thead>
<tr>
<th>TABLE IV: Malware detection rate comparison.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware name</td>
</tr>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Android.Malware.at_plapka</td>
</tr>
<tr>
<td>Android.Troj.at_fonefee.b</td>
</tr>
<tr>
<td>brainest</td>
</tr>
<tr>
<td>ghostpush</td>
</tr>
<tr>
<td>candy_corn</td>
</tr>
</tbody>
</table>

* All commercial packers can successfully detect it as malware.
Findings - Evolution

- Evolution: Android packers have been evolving very fast in the last few years.
- Number of unpacking layers.

Fig. 5: Layer distribution.
Conclusion

• We conduct the first large-scale study on Android packers
  • commercial packers have been increasingly abused
  • severe security issues are introduced by packers
  • Android Packers are quickly evolving
• We design and implement DroidUnpack and it will be released later
THANK YOU!!
Reference


DroidUnpack System - Discussion

- Data Compression and Encoding
- Supporting Android versions
- Emulation Detection
Motivation

• Why Android?

• Android is popular + it has design issues.
  • iOS enforces code signing to prohibit app from any modification since it was last signed.
  • Android allows the code to be modified even after installation.
DroidUnpack System

• Code Behavior Analysis
  • Hidden OAT/DEX code extraction

• Self-modifying code detection

• Multi-layer unpacking detection

• Java native interface inspection