DABiD
The Powerful Interactive Android Debugger for Android Malware Analysis
Reversing Android application

- **Static Analysis**
  - Analyze intent of application by decoding DEX (Dalvik Executable) into readable bytecode
    - ex) Apktool, JEB

- **Dynamic Analysis**
  - Monitor behavior of android application at runtime
    - ex) DroidBox, Mobile Sandbox, Anubis etc.
  - Conduct step by step debugging with disassembled Dalvik executable code
    - ex) SmaliDebugging, IDAPro
Smali Debugging

- Use apktool and NetBeans (Java IDE) in combination
  - Apktool: disassemble DEX and repackage app in debug mode
  - Java IDE: support step by step debugging
**Dalvik Executable Debugging**

- **IDA Pro Debugging**
  - Supports dalvik debugging from version 6.6
  - Similar to Smali Debugging but use own DEX disassembler
# Dalvik Executable Debugging

## Smali Debugging VS. IDA Pro Debugging

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing</th>
<th>Dalvik Executable Disassemble</th>
<th>Debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application Modification</td>
<td>Debugging Settings</td>
<td>Disassembler</td>
</tr>
<tr>
<td>Smali Debugging</td>
<td>modified as debuggable</td>
<td>Jdwp socket host &amp; port</td>
<td>Smali</td>
</tr>
<tr>
<td>IDA Pro</td>
<td></td>
<td>Package &amp; launchable activity name</td>
<td>IDA Pro</td>
</tr>
</tbody>
</table>

What if...The dex I am debugging is not the one running??
Challenges in Android Analysis: Modification of DEX bytes at runtime

- BlueBox Security verified tempering davik bytecode during runtime is possible
  - Load library and execute function which write bytes into memory where dalvik executable is loaded
    - Find codeltem of “add()” method from DEX loaded in memory
    - Write bytes into codeltem of “add()” method

```c
void *fastcall search(unsigned int a1)
{
    ...
    v1 = a1;
    v2 = syscall(39);
    v3 = v1 - v1 % v2;
    do
    {
        v3 += v2;
        v4 = v3 + 40;
    }
    while (!findmagic((const void *)&(v3 + 40))); // instruction of add

    v5 = getStrIdx(v3 + 40, "Ljava/lang/String;", 0x12u);
    v6 = getStrIdx(v4, "add", 3u);
    v7 = getTypeIdx(v4, v5);
    v8 = getClassIdx(v4, v7);
    v9 = getMethodIdx(v4, v6, v7);
    v10 = getCodeIdx(v4, v6, v9);
    v11 = (v10 + 16); // instruction of add

    return memcpy(v11, inject, 0x10u);
}
```

Diagram:
- “Ljava/lang/String;” — “add”
  - ClassIdx
  - MethodIdx
  - CodeIdx
  - Codeltem
  - Find codeltem of “add()”
  - Write “inject” bytes into codeltem
Challenges in Android Analysis: Dynamic DEX Loading

- GoogleAppsToy malware load encrypted DEX at runtime
  - Analysts can obtain classes.dex from APK and conduct static or dynamic analysis on classes.dex
  - But, classes.dex from APK has no malicious actions
    - Only decrypt and dynamic loading routine exist in classes.dex
    - No way to debug malicious code....

```
File System
```
```
classes.dex
Loading Stub
```
```
Encrypted dex (Asset)
```
```
Process
```
```
classes.dex
LoadingStub
```
```
Dynamically loaded dex
Malcode
```
Challenges in Android Debugging

- DEX (Dalvik Executable) can be different in memory

Classes.dex in APK ≈ Classes.dex in Dalvik-cache (optimized) ≠ Dex in Memory

Analysis Target

Analysis with current analyzers might be useless..
DABiD: Dynamic Android Binary Debugger
Goal

• To develop android debugger which is able to debug “the same DEX” running on memory

• To make android debugging more effective and convenient for analysts
Our Approach (1/2)

• To develop android debugger which is able to debug “the same DEX” running on memory
  — Monitor dynamic changes in memory and reflect them to debugger
    • Self modification of DEX bytes in memory
    • Dynamic DEX loading

• To make android debugging more effective and convenient for analysts
Our Approach (2/2)

- To develop android debugger which is able to debug “the same DEX” running on memory
- To make android debugging more effective and convenient for analysts
  - Provide advanced debugging features
    - Code update by analyst
    - Register value acquisition
  - Automate bothersome settings for android debugging
DABiD - Overview

- Resembles java debugger structure
  - DEX disassembler: disassemble DEX from both apk file and memory
  - Debug Event Handler: create and handle debugging event from JDWP
- But, JDWP has limitations...
DABiD - Overview

- Make our service module reside in application process
  - Notify supervision results of dynamic changes in memory
  - Give a control over the application
Monitoring Dynamic Changes
: Self modification of DEX bytes

- DABiD Service module detects that memory write function call is made and alarm debugger when event happens

- Analysts is now able to analyze hidden bytes
Monitoring Dynamic Changes: Dynamic DEX loading

- **DABiD Service module** find and dump new DEX bytes in memory

- **With DABiD**, Analysts no longer need to
  - Find the location of hidden or newly downloaded DEXs
  - Conduct static analysis for new DEXs by pulling them to local
Advanced Debugging Feature: Code update by Analyst

• Analysts can modify bytecodes on the fly
  — Analysts can input bytes from DABiD
  — DABiD service module writes input bytes to proper location in memory
  — Debuggee runs with modified bytes

• Analysts are able to
  — Force to execute code
  — Skip code part should not be executed to continue analysis
**Advanced Debugging Feature**
: Register value acquisition

- JDI provides register values only with debug symbols
  - But, Not all registers have debug symbols

<table>
<thead>
<tr>
<th>Java</th>
<th>Bytecode</th>
<th>DebugSymbol</th>
</tr>
</thead>
</table>
| `int a = 3; Log.d("Info", "a : " + a);` | `.local name :v0' type: int
  - const/4
  - const-string
  - new-instance
  - const-string
  - Invoke-direct
  - Invoke-virtual ...` | `v0` = int |

- Modify JDI to get values of registers
  - Get register value using slot number
  - Eliminate evaluation check whether the register is visible variable or not
  - Cast value with type information by emulating bytecodes in DEX disassembler

<table>
<thead>
<tr>
<th>Java</th>
<th>Bytecode</th>
<th>DebugSymbol</th>
</tr>
</thead>
</table>
| `int a = 3; Log.d("Info", "a : " + a);` | `.local name :v0' type: int
  - const/4
  - const-string
  - new-instance
  - const-string
  - Invoke-direct
  - Invoke-virtual ...` | `v0` = int

Analyzed type info
Automation of debugging setting

• DABiD automates followings to aid debugging
  — Transform application into debuggable
  — Install and start application
  — Set jdwp socket connection
  — Set breakpoints at the starting point of application
DEMO
Future work

- Debugging for Android Runtime (ART)
  - Support ART features
  - Resolve Code Protections on ART

- Code coverage
  - Support native code debug included in APK

- Anti-Debugging
  - Counter anti-debugging techniques