StaDynA: Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android Apps

Yury Zhauniarovich, Maqsood Ahmad, Olga Gadyatskaya, Bruno Crispo, Fabio Massacci
yury.zhauniarovich, maqsood.ahmad, bruno.crispo, fabio.massacci@unitn.it
olga.gadyatskaya@uni.lu

University of Trento
SnT, University Of Luxembourg
Analysis Types

- **Static analysis** – is the analysis of applications which is performed without the actual execution of an application

- **Dynamic analysis** – is the analysis which is performed by executing an application in real or emulated environments
Dynamic Code Updates*

1. Dynamic Class Loading (DCL)
2. Reflection

*S. Poeplau et al. “Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications”. In Proc. Of NDSS’14
Motivation

- In Android, code loaded dynamically has the same privileges as original
- Static analyzers cannot fully inspect an app in the presence of dynamic code update features (AndroGuard, FlowDroid, etc.)
- Heavily used by malware to conceal malicious behavior
- Used in real applications to bypass Android limitations
Reflection and DCL Usage

- **Google Play:**
  - analyzed 13863 apps
  - 19% contain DCL calls
  - 88% use reflection

- **Third-party markets:**
  - analyzed 14283 apps from 6 markets
  - 6% contain DCL calls (F-Droid: 1%)
  - 74% use reflection (F-Droid: 57%)

- **Malware dataset:**
  - 1260 samples analyzed
  - 20% contain DCL calls
  - 81% use reflection
```java
[com.sec.android.providers.drm.Doctype]
public static Object b(File paramFile, String paramString1, String paramString2, Object[]
paramArrayOfObject)
{
    String str3;
    if (paramFile == null) {
        String str1 = a.getFileDir().getAbsolutePath();
        // get the name of the file to be loaded
        //9CkOrC32uI327WBD7n_---> /anserverv.db
        String str2 = Xnins.d("9CkOrC32uI327WBD7n_");
        str3 = str1.concat(str2);
    }
    for (File localFile = new File(str3); ; localFile = paramFile) {
        String str4 = localFile.getAbsolutePath();
        String str5 = a.getFileDir().getAbsolutePath();
        ClassLoader localClassLoader = a.getClassLoader().getParent();
        // get the class specified by "paramString1" from anserverv.db
        Class localClass = new DexClassLoader(str4, str5, null, localClassLoader).loadClass(
            paramString1);
        Class[] arrayOfClass = new Class[5];
        arrayOfClass[0] = Context.class;
        arrayOfClass[1] = Intent.class;
        arrayOfClass[2] = BroadcastReceiver.class;
        arrayOfClass[3] = FileDescriptor.class;
        arrayOfClass[4] = String.class;
        // get the method specified by "paramString2"
        Method localMethod = localClass.getMethod(paramString2, arrayOfClass);
        // create new instance of the class
        Object localObject = localClass.newInstance();
        // call the corresponding method with arguments in array "paramArrayOfObject"
        return localMethod.invoke(localObject, paramArrayOfObject);
    }
}
```
Problem: Dynamic Code Updates

**Issue:** How to analyze Android apps in the presence of

- reflection calls,
  - detect the name of the called function/class
- dynamic class loading?
  - download and analyze the loaded code

- Method Call Graph (MCG) is a directed graph showing the calling relationships between methods in a computer program
StaDynA: Idea

- Apps with Dynamic Code Update features expose their dynamic behavior at runtime

- IDEA: combine static and dynamic analysis techniques to detect and explore Dynamic Code Update features
StaDynA: Overview
StaDynA: Approach

- Find API calls responsible for reflection and DCL at static time (we name the methods calling these API functions as **Methods of Interest (MOI)**)

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Prot.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dynamic class loading</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Ldalvik/system/PathClassLoader</code>;</td>
<td><code>&lt;init&gt;</code></td>
<td>.</td>
</tr>
<tr>
<td><code>Ldalvik/system/DexClassLoader</code>;</td>
<td><code>&lt;init&gt;</code></td>
<td>.</td>
</tr>
<tr>
<td><code>Ldalvik/system/DexFile</code>;</td>
<td><code>&lt;init&gt;</code></td>
<td>.</td>
</tr>
<tr>
<td><code>Ldalvik/system/DexFile</code>;</td>
<td><code>loadDex</code></td>
<td>.</td>
</tr>
<tr>
<td><strong>Class instance creation through reflection</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Ljava/lang/Class</code>;</td>
<td><code>newInstance</code></td>
<td>.</td>
</tr>
<tr>
<td><code>Ljava/lang/reflect/Constructor</code>;</td>
<td><code>newInstance</code></td>
<td>.</td>
</tr>
<tr>
<td><strong>Method invocation through reflection</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Ljava/lang/reflect/Method</code>;</td>
<td><code>invoke</code></td>
<td>.</td>
</tr>
</tbody>
</table>

- Analyze their behavior at runtime
StaDynA: Workflow

USER

APK

SERVER

Get Code

Analyze Code
Build MCG

Analyze DCL Event

CLIENT

DCL triggered

Triggering Solution

Stop Analysis?

yes

no

Reflection triggered

Analysis

Analyse call
Add edge

Analyze Reflection Event

App Testing

a

b
c
d
e

f
g

1

2

3

4

5

6

7

8

9

10

11
StaDynA: Features

- Stores and analyzes the code loaded dynamically
- Builds MCG of the app including the information obtained at runtime
- Discovers at runtime the qualifiers of the methods/constructors called through reflection
- Discovers suspicious behavior patterns
StaDynA: Evaluation

- Dataset:
  - 5 benign (FlappyBird, Norton AV, Avast AV, Viber, Floating Image)
  - 5 malicious (FakeNotify.B, AnserverBot, BaseBridge, DroidKungFu4, SMSSend)

- The dataset is small:
  - StaDynA requires manual triggering

- Evaluation parameters:
  - the increase of the MCG
  - coverage of the MOI detected in the application
  - discovered suspicious patterns
## Evaluation: MCG Increase

<table>
<thead>
<tr>
<th>Apps</th>
<th>Nodes</th>
<th></th>
<th>Edges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial</td>
<td>Final</td>
<td>Initial</td>
<td>Final</td>
</tr>
<tr>
<td><strong>Benign Applications</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FlappyBird</td>
<td>8592</td>
<td>8614</td>
<td>11014</td>
<td>11031</td>
</tr>
<tr>
<td>Norton AV</td>
<td>42886</td>
<td>55372</td>
<td>65960</td>
<td>85665</td>
</tr>
<tr>
<td>Avast AV</td>
<td>31317</td>
<td>32363</td>
<td>43554</td>
<td>44956</td>
</tr>
<tr>
<td>Viber</td>
<td>42536</td>
<td>46312</td>
<td>60078</td>
<td>65627</td>
</tr>
<tr>
<td>ImageView</td>
<td>5708</td>
<td>5713</td>
<td>6488</td>
<td>6496</td>
</tr>
<tr>
<td><strong>Malicious Applications</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FakeNotify.B</td>
<td>148</td>
<td>171</td>
<td>137</td>
<td>191</td>
</tr>
<tr>
<td>AnserverBot</td>
<td>1006</td>
<td>1614</td>
<td>1138</td>
<td>2093</td>
</tr>
<tr>
<td>BaseBridge</td>
<td>1172</td>
<td>1780</td>
<td>1364</td>
<td>2333</td>
</tr>
<tr>
<td>DroidKungFu4</td>
<td>1550</td>
<td>21168</td>
<td>1779</td>
<td>23589</td>
</tr>
<tr>
<td>SMSSend</td>
<td>431</td>
<td>537</td>
<td>826</td>
<td>951</td>
</tr>
</tbody>
</table>
### Evaluation: Coverage

<table>
<thead>
<tr>
<th>Apps</th>
<th>Refl. Invoke</th>
<th>Refl. NewInstance</th>
<th>DCL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Found (Init.)</td>
<td>Triggered</td>
<td>Found (Init.)</td>
</tr>
<tr>
<td>FlappyBird</td>
<td>11 (10)</td>
<td>6</td>
<td>6 (6)</td>
</tr>
<tr>
<td>Norton AV</td>
<td>137 (18)</td>
<td>5</td>
<td>12 (8)</td>
</tr>
<tr>
<td>Avast AV</td>
<td>42 (42)</td>
<td>6</td>
<td>19 (19)</td>
</tr>
<tr>
<td>Viber</td>
<td>107 (101)</td>
<td>26</td>
<td>47 (21)</td>
</tr>
<tr>
<td>ImageView</td>
<td>6 (6)</td>
<td>5</td>
<td>2 (2)</td>
</tr>
<tr>
<td><strong>Benign Applications</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FakeNotify.B</strong></td>
<td>68 (68)</td>
<td>68</td>
<td>9 (9)</td>
</tr>
<tr>
<td>AnserverBct</td>
<td>4 (4)</td>
<td>1</td>
<td>5 (4)</td>
</tr>
<tr>
<td>BaseBridge</td>
<td>5 (5)</td>
<td>1</td>
<td>3 (2)</td>
</tr>
<tr>
<td>DroidKungFu4</td>
<td>13 (9)</td>
<td>1</td>
<td>6 (4)</td>
</tr>
<tr>
<td>SMSSend</td>
<td>193 (193)</td>
<td>128</td>
<td>1 (1)</td>
</tr>
</tbody>
</table>
# Evaluation: Suspicious Patterns

## Benign Applications

<table>
<thead>
<tr>
<th>Norton AV</th>
<th>WRITE_SETTINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>READ_PHONE_STATE</td>
</tr>
<tr>
<td></td>
<td>INTERNET</td>
</tr>
<tr>
<td></td>
<td>WRITE_SYNC_SETTINGS</td>
</tr>
<tr>
<td></td>
<td>GET_TASKS</td>
</tr>
<tr>
<td>Avast AV</td>
<td>INTERNET</td>
</tr>
<tr>
<td>Viber</td>
<td>READ_PHONE_STATE</td>
</tr>
<tr>
<td></td>
<td>BLUETOOTH</td>
</tr>
<tr>
<td></td>
<td>INTERNET</td>
</tr>
</tbody>
</table>

## Malware

<table>
<thead>
<tr>
<th>Malware</th>
<th>SEND_SMS</th>
<th>INTERNET</th>
</tr>
</thead>
<tbody>
<tr>
<td>FakeNotify.B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AnserverBot</td>
<td></td>
<td>READ_PHONE_STATE</td>
</tr>
<tr>
<td>BaseBridge</td>
<td></td>
<td>READ_PHONE_STATE</td>
</tr>
<tr>
<td>DroidKungFu4</td>
<td>CHANGE_NETWORK_STATE</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>ACCESS_COARSE_LOCATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLUETOOTH</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>INTERNET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLUETOOTH_ADMIN</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>WRITE_SETTINGS</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>SET_TIME_ZONE</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>WRITE_SYNC_SETTINGS</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>READ_PHONE_STATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANGE_WIFI_STATE</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>MODIFY_AUDIO_SETTINGS</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>MOUNT_UNMOUNT_FILESYSTEMS</td>
<td>✓</td>
</tr>
<tr>
<td>SMSSend</td>
<td>READ_PHONE_STATE</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>SEND_SMS</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **Access to the functionality protected with dangerous permissions from the loaded code**

- **Ticks show that the usage of the corresponding permission has not been found in the initial app file (over-privileged apps)**
FakeNotify.B before StaDynA
FakeNotify.B after StaDynA
StaDynA: Issues

- Manual triggering
- Resolution of all reflection targets is done at runtime
- The information obtained during different runs is not merged
- No separation according to the name of the package (UID is used instead)
- Not all types of dynamic code updates have been covered
StaDynA: Summary

- Dynamic code updates is a serious problem for Android
  - the code loaded dynamically has the same privileges as the original application

- We proposed an approach that facilitates the analysis of apps in the presence of reflection and DCL
  - discovers at runtime the qualifiers of the methods/constructors called through reflection
  - stores and analyzes code loaded dynamically
  - builds MCG of the app including the information obtained at runtime
  - discovers suspicious behavior patterns

- Open-source: https://github.com/zyrikby/StaDynA
BACKGROUND SLIDES
StaDynA: Main Function

Protocol 4 App analysis main function algorithm

1: function perform_analysis(inputApkPath, resultsDirPath)
2:     makeAnalysis(inputApkPath)
3:     if !containsMethodsToAnalyze() then
4:         performInfoSave(resultsDirPath)
5:     return
6: end if
7: dev ← getDeviceForAnalysis()
8: package_name ← get_package_name(inputApkPath)
9: dev.install_package(inputApkPath)
10: uid ← dev.get_package_uid(package_name)
11: messages ← dev.getLogcatMessages(uid)
12: loop
13:     msg ← dequeue(messages)
14:     analyseStadyna_Msg(msg)
15:     if finishAnalysis then
16:         performInfoSave(resultsDirPath)
17:     return
18: end if
19: end loop
20: end function
Protocol 6 The algorithm for analysis of the reflection invoke message

1: function PROCESS_REF_invokeMsg(message)
2:     cls ← message.get(JSON_CLASS)
3:     method ← message.get(JSON_METHOD)
4:     prototype ← message.get(JSON_PROTO)
5:     stack ← message.get(JSON_STACK)
6:     invDstFrCl ← (class, method, prototype)
7:     invPosInStack ← findFirstInvokePos(stack)
8:     thrMtd ← stack[invPosInStack]
9:     invSrcFrStack ← stack[invPosInStack + 1]
10:    for all invPathFrSrcs ∈ sources_invoke do
11:        invSrcFrSrcs ← invPathFrSrcs[0]
12:        if invSrcFrSrcs ≠ invSrcFrStack then
13:           continue
14:        end if
15:        addInvPathToMCG(invSrcFrSrcs, thrMtd, invDstFrCl)
16:        if invPathFrSrcs ∈ uncovered_invoke then
17:           uncovered_invoke.remove(invPathFrSrcs)
18:        end if
19:    end for
20: end function
Analysis of DCL Event

Protocol 7 The algorithm for analysis of the DCL message

1: function PROCESSDEXLOADMSG(message)
2: source ← message.get(JSON_DEX_SOURCE)
3: stack ← message.get(JSON_STACK)
4: newFile ← dev.get_file(source)
5: newPath ← processNewFile(newFile)
6: dlPathFrStack = getDLPathFrStack(stack)
7: if dlPathFrStack then
8: srcFromStack ← dlPathFrStack[0]
9: thrMtd ← dlPathFrStack[1]
10: if dlPathFrStack ∈ uncovered_dexload then
11: uncovered_dexload.remove(dlPathFrStack)
12: end if
13: addDLPathToMCG(srcFromStack, thrMtd, newPath)
14: if !fileAnalysed(newPath) then
15: makeAnalysis(newPath)
16: end if
17: return
18: end if
19: end function