Androsia

Securing 'data in process' for your Android Apps
C:\>whoami

- Samit Anwer
- Product Security Team @Citrix R&D India Pvt Ltd
- Web/Mobile App Security Enthusiast
- Speaker:
  - AppSec USA (Orlando, USA) 2017,
  - c0c0n (Cochin, India) 2017,
  - CodeBlue (Tokyo, Japan) 2017,
  - IEEE Services (Anchorage, Alaska) 2014,
  - ACM MobileSOFT, ICSE (Hyderabad, India) 2014,
  - IEEE CloudCom (Bristol, UK) 2013

Email: samit.anwer@gmail.com, Twitter: @samitanwer1, LinkedIn: https://www.linkedin.com/in/samit-anwer-ba47a85b/
Which one is the most difficult to protect?

1. Data at Rest
2. Data in Process
3. Data in Motion
Motivation
Want to ensure object’s content gets cleared?

Myth:

THE GARBAGE COLLECTOR
WILL HANDLE IT

Reality:

Unreachable objects ➔ Garbage
Forgotten References ➔ Memory Leak
Reachable objects
Garbage Collection Roots

Ref: https://www.dynatrace.com/resources/ebooks/javabook/how-garbage-collection-works/
Resetting StringBuilder objects

- Reachable, unused StringBuilder objects may contain sensitive information
- A heap dump will reveal the sensitive info
- Don’t just rely on GC to clear sensitive content
- Destroy by overwriting all critical data

java.security.* falls short

**Class KeyStore.PasswordProtection**

java.lang.Object
   | java.security.KeyStore.PasswordProtection

All Implemented Interfaces:
KeyStoreProtectionParameter, Destroyable

Enclosing class:
KeyStore

```java
public static class KeyStore.PasswordProtection
extends Object
implements KeyStoreProtectionParameter, Destroyable

A password-based implementation of ProtectionParameter.
```

```java
public void destroy()
    throws DestroyFailedException

Clears the password.
```

Specified by:
destroy in interface Destroyable

Throws:
DestroyFailedException - if this method was unable to clear the password

Ref: https://docs.oracle.com/javase/7/docs/api/java/security/KeyStore>PasswordProtection.html
How does Androsia help?
• Androsia determines last use of objects at a whole program level
  • A summary based inter-procedural data-flow analysis
• Androsia instruments bytecode to clear memory content of objects
### Eclipse Memory Analyzer

#### Heap Dump - Before Instrumentation

- User
- com.example.getset
- class java.lang.Class @ 0xa4ce11e8
- Object
- dalvik.system.PathClassLoader @ 0xca4fd490
- 8 (shallow size)

#### Heap Dump - After Instrumentation

<table>
<thead>
<tr>
<th>Static</th>
<th>Attributes</th>
<th>Class Hierarchy</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>StaticOverhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ref</td>
<td>static_secret</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The red circles highlight the columns to focus on.
Overview

Androsia

1. Shares source OR APK
2. Unpack
3. Convert
4. Transform/Instrument
5. Convert
6. Repack & Sign APK or Provide Analysis Results
7. Transformed APK/Results
Framework behind Androsia
Static Code analysis using Soot

• Soot - framework for Java (bytecode), enables development of static analysis tools
  • Provides three-address code called Jimple
  • Supports implementing dataflow analyses:
    • Intra-procedural
    • Inter-procedural

• Soot was missing a Dalvik to Jimple transformation module
  • and then came Dexpler

Soot Workflow
FlowDroid

- Android apps don’t have a main method

- FlowDroid generates `dummyMainMethod()`

- Models Android’s lifecycle methods & callbacks

Further reading:
Instrumenting Android Apps with Soot, [http://www.bodden.de/2013/01/08/soot-android-instrumentation/](http://www.bodden.de/2013/01/08/soot-android-instrumentation/)

Img. ref: [https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle](https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle)
SB Objects – In what scopes can they exist?

Local variable

```java
public void foo(){
    SB x, y, z;
    [x = new SB("s3cr3t");]¹
    [y = new SB("p@55w0rd");]²
    [if(y.length() < x.length()){
        [z = y;]⁴
    } else{
        [z = y.append("007");]⁵
    }]³
}
```

Static Field

```java
class MyClass{
    static SB x;
    public static void foo(){
        SB y, z;
        [x = new SB("s3cr3t");]¹
        [y = new SB("p@55w0rd");]²
        [if(y.length() < x.length()) {
            Abbrev.
            [z = y;]⁴
            SB: StringBuilder
        } else{
            bar();⁵
        }]
    }
    public static void bar(){
        System.out.println(StaticSB.x);
    }
}
```

Abbrev.
SB: StringBuilder
public static void foo(){
    MyInstanceFieldSB obj = new MyInstanceFieldSB();
    SB str = new SB();

    obj.setSBx(str)

    S.O.P(obj.getSBx());
}

class MyInstanceFieldSB {
    private SB x;

    public SB getSBx(){
        return x;
    }

    public SB setSBx(SB str){
        x = str;
    }
}
public class MainActivity {
    protected void onCreate(Bundle b) {
        User.static_secret = new SB("p@55w0rd");
        CheckStatic cs = new CheckStatic();
        cs.useStaticField();
    }
}

public class User {
    public static SB static_secret;
}

public class CheckStatic {
    public void useStaticField() {
        S.O.P(User.static_secret);
        bar();
    }
    public void bar() {
        S.O.P(User.static_secret);
    }
}

But life is not always so simple - There can be loops
Approach
What’s there in a line of code?

• What data are we interested in?

Next few slides:
• What is live variable analysis?

• How to compute Summary for every method?
  e.g. Summary(foo) = ( x, if y.length() < x.length() )

  ➢ Step 1: Compute def-use set for every statement
  ➢ Step 2: Compute \( L_{\text{entry}} \) & \( L_{\text{exit}} \) set for every statement
  ➢ \( L_{\text{entry}} \) & \( L_{\text{exit}} \) → Last Usage Point (LUP) for Local / Static Field Ref. (SFR) within a method → Summary

• How to use summaries to compute LUP for a SFR at a whole program level?
Live Variable Analysis

• LV analysis determines
  • For each statement, which variables \textit{must} have a \textit{subsequent USE} prior to next definition of that variable

Last Usage Point of a var \(\cong\) Last stmt where that var was live

Abbrev.
SB: StringBuilder

public void foo(){
SB x, y, z;

[x = new SB(“s3cr3t”);]¹
[y = new SB(“p@55w0rd”);]²
[x = new SB(“hello”);]³

[if(y.length() < x.length()) {
  [z = y;]⁵
} else{
  [z = y.append(“007”);]⁶
}]⁴

[x = z;]⁷
}
1. Compute def-use Set

- **def set**: variables defined in the statement
- **use Set**: variables evaluated/used in the statement

```java
public void foo(){
    SB x, y, z;
    [x = new SB("s3cr3t");]¹
    [y = new SB("p@55w0rd");]²
    [x = new SB("hello");]³
    [if(y.length() < x.length()) {
        [z = y;]⁵
    } else{
        [z = y.append("007");]⁶
    }]⁴
    [x = z;]⁷
}
```

Abbrev.
SB: StringBuilder
1. Compute def-use Set (cntd.)

<table>
<thead>
<tr>
<th></th>
<th>$\text{def}(l)$</th>
<th>$\text{use}(l)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{x}</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>{y}</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>{x}</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>{x, y}</td>
</tr>
<tr>
<td>5</td>
<td>{z}</td>
<td>{y}</td>
</tr>
<tr>
<td>6</td>
<td>{z}</td>
<td>{y}</td>
</tr>
<tr>
<td>7</td>
<td>{x}</td>
<td>{z}</td>
</tr>
</tbody>
</table>

SB $x, y, z$;

$[x = \text{new } \text{SB}(\text{"s3cr3t"});]^{1}$
$[y = \text{new } \text{SB}(\text{"p@55w0rd"});]^{2}$
$[x = \text{new } \text{SB}(\text{"hello"});]^{3}$

$\text{if}(y.\text{length}() < x.\text{length}()){\ }
\hspace{1cm}[z = y;]^{5}$

$\text{else}{\ }
\hspace{1cm}[z = y.\text{append}(\text{"007"});]^{6}$

$}[]^{4}$
$[x = z;]^{7}$
2. Compute $LV_{\text{entry}} (l)$ & $LV_{\text{exit}} (l)$

• Hence the flow equations can be expressed using the two functions:

$$LV_{\text{exit}} (l) = \emptyset$$

$$\cup_{s \in \text{succ} [l]} \{LV_{\text{entry}} (s)\}, \text{ if } l = b_{\text{last}}$$

$$LV_{\text{entry}} (l) = (LV_{\text{exit}} (l) - \text{def}(l)) \cup \text{use}(l)$$
3: Compute $LV_{\text{entry}}(l)$ & $LV_{\text{exit}}(l)$

$$LV_{\text{entry}}(l) = (LV_{\text{exit}}(l) - \text{def}(l)) \cup \text{use}(l)$$

<table>
<thead>
<tr>
<th>I</th>
<th>def(l)</th>
<th>use(l)</th>
<th>I</th>
<th>$LV_{\text{entry}}(l)$</th>
<th>$LV_{\text{exit}}(l)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{x}</td>
<td>$\emptyset$</td>
<td>1</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>{y}</td>
<td>$\emptyset$</td>
<td>2</td>
<td>$\emptyset$</td>
<td>{y}</td>
</tr>
<tr>
<td>3</td>
<td>{x}</td>
<td>$\emptyset$</td>
<td>3</td>
<td>{y}</td>
<td>{x, y}</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>{x, y}</td>
<td>4</td>
<td>{x, y}</td>
<td>{y}</td>
</tr>
<tr>
<td>5</td>
<td>{z}</td>
<td>{y}</td>
<td>5</td>
<td>{y}</td>
<td>{z}</td>
</tr>
<tr>
<td>6</td>
<td>{z}</td>
<td>{y}</td>
<td>6</td>
<td>{y}</td>
<td>{z}</td>
</tr>
<tr>
<td>7</td>
<td>{x}</td>
<td>{z}</td>
<td>7</td>
<td>{z}</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

**Summary(\text{foo}) = \{ \text{Local, LUP( Local / Aliases )} \} OR \{ \text{StaticFieldRef, LUP( SFR / Aliases )} \}**

```java
public void foo()
{
    SB x; SB y; SB z;
    [x = new SB("s3cr3t");]¹
    [y = new SB("p@55w0rd");]²
    [x = new SB("hello");]³
    if(y.length() < x.length()) {
        [z = y;]⁵
        [z = y.append("007");]⁶
    } else{
        [z = y.append("007");]⁶
    }
    [x = z;]⁷
}
```

1 $LV_{\text{entry}}(2)$ $\{ \emptyset \}$
2 $LV_{\text{exit}}(2)$ $\{ y \}$
3 $LV_{\text{entry}}(3)$ $\{ y \}$
4 $LV_{\text{exit}}(3)$ $\{ x, y \}$
5 $LV_{\text{entry}}(5)$ $\{ y \}$
6 $LV_{\text{exit}}(5)$ $\{ z \}$
7 $LV_{\text{entry}}(7)$ $\{ z \}$
8 $LV_{\text{exit}}(7)$ $\emptyset$
Summary is computed in reverse topological order

**public void foo()**{

**A1**
**A2**
**A3** **bar();**
**A4**
**A5**
}

**public void bar()**{

**B1**
**B2**

**B3** **baz();**

**B4**

**B5** **sfr used**

**C1**

**C2**

**C3**

**C4** **sfr used**

**C5**

**C6**

**LV_{exit}(A3) = LV_{entry}(A4) = \{\emptyset\}**

**LV_{exit}(B3) = LV_{entry}(B4) = \{sfr, B5\}**

**LV_{entry}(B5) = \{sfr, B5\}**

**Summ(bar) = \{sfr, B5\}**

**Summ(baz) = \{sfr, C4\}**

Program level last use for “sfr” happens at:

**\{baz, (sfr,C4)\}**

**\{ bar, (sfr,B5) \}**

**Summ(bar) = \{sfr, \emptyset\}**
What is live variable analysis? ✓

How to compute Summary for every method?
  e.g. \( \text{Summary}(\text{foo}) = (x, \text{if}(y.\text{length}() < x.\text{length}())) \)

- Step 1: Compute def-use set for every statement ✓
- Step 2: Compute \( L_{\text{entry}} \) & \( L_{\text{exit}} \) set for every statement ✓
- \( L_{\text{entry}} \) & \( L_{\text{exit}} \) → Last Usage Point (LUP) for Local / Static Field Ref. (SFR) within a method → Summary ✓

How to use summaries to compute LUP for a SFR at a whole program level? ✓
Instance Field Approach

• Mark all **classes** which have **StringBuilder Instance Field/s**
• Find their **object instances**
• Track **Last Usage of object instances & their aliases** instead of SB Fields
• Add **reset method/s** to respective class
• Mark all classes which have StringBuilder Instance Field/s
• Find their object instances
• Track Last Usage of object instances & their aliases instead of SB Fields
• Add reset method/s to respective class
Test Suite development
CI/CD adoption

Get in touch & contribute:

I will be releasing the tool and documentation at the end of the conference!

Twitter: @samitanwer1,
Email: samit.anwer@gmail.com,
LinkedIn: https://www.linkedin.com/in/samit-anwer-ba47a85b/

2. Instrumenting Android Apps with Soot, [http://www.bodden.de/2013/01/08/soot-android-instrumentation](http://www.bodden.de/2013/01/08/soot-android-instrumentation)


4. Precise Interprocedural Dataflow Analysis via Graph Reachability, [https://courses.cs.washington.edu/courses/cse590md/01sp/w1l2.pdf](https://courses.cs.washington.edu/courses/cse590md/01sp/w1l2.pdf)

5. Slides by Matthew B. Dwyer and Robby, University of Nebraska-Lincoln, Kansas State University
public int foo(java.lang.String) {
    // [local defs]
    r0 := @this; // IdentityStmt
    r1 := @parameter0;

    if r1 != null goto label0; // IfStmt
    $i0 = r1.length(); // AssignStmt
    r1.toUpperCase(); // InvokeStmt
    return $i0; // ReturnStmt

    label0:
        return 2;
}