
# 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

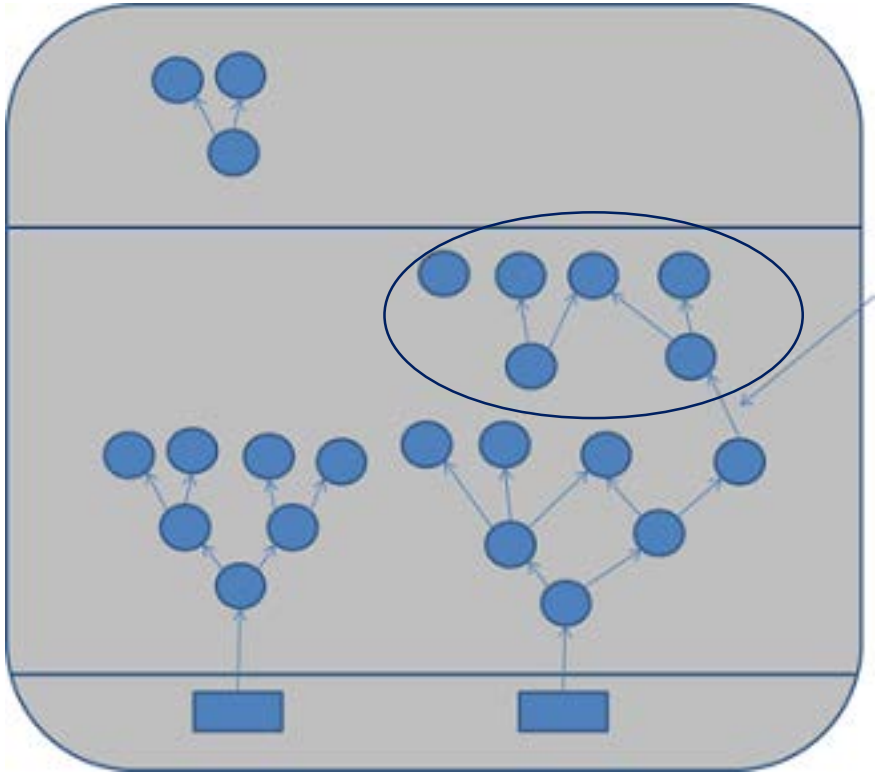
Data in Motion

# Motivation

# Want to ensure object's content gets cleared?

**Myth:**


**Reality:**



Unreachable objects → Garbage

Forgotten References → Memory Leak

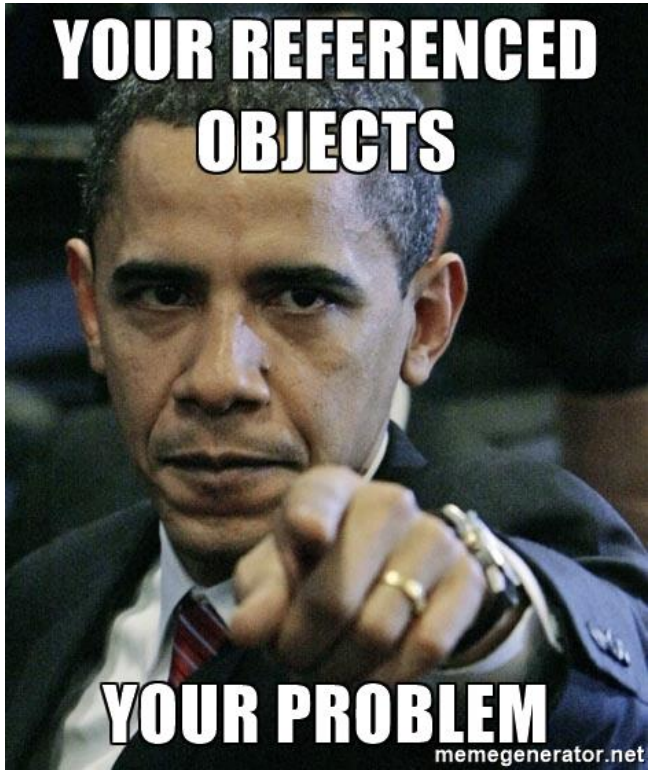
Reachable objects

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




Ref: https://www.pentestpartners.com/security-blog/how-to-extract-sensitive-plaintext-data-from-android-memory

# java.security.* falls short

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.security

## Class KeyStore.PasswordProtection

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

**All Implemented Interfaces:**

KeyStore.ProtectionParameter, Destroyable

**Enclosing class:**

KeyStore

---

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

A password-based implementation of ProtectionParameter.

**destroy**

```
public void destroy()
            throws DestroyFailedException
```

Clears the password.

**Specified by:**

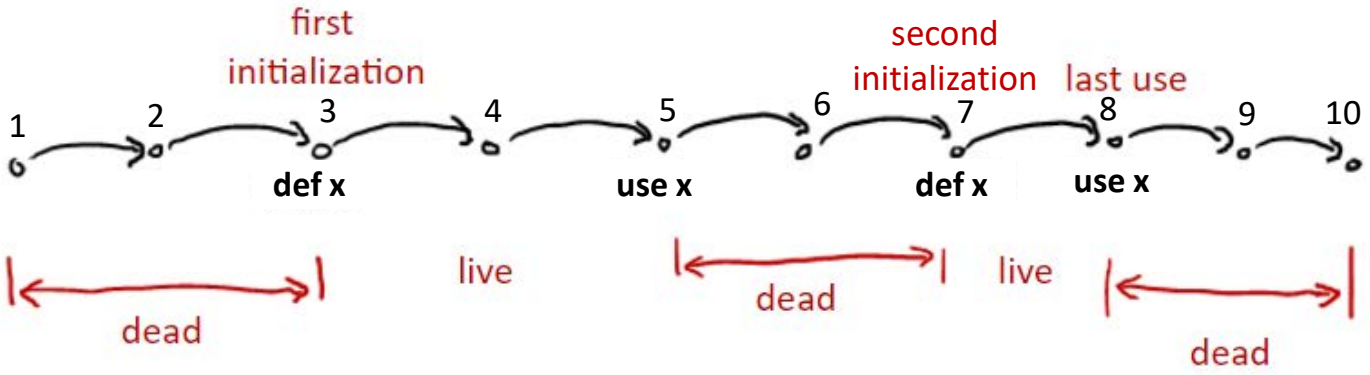
destroy in interface Destroyable

**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?

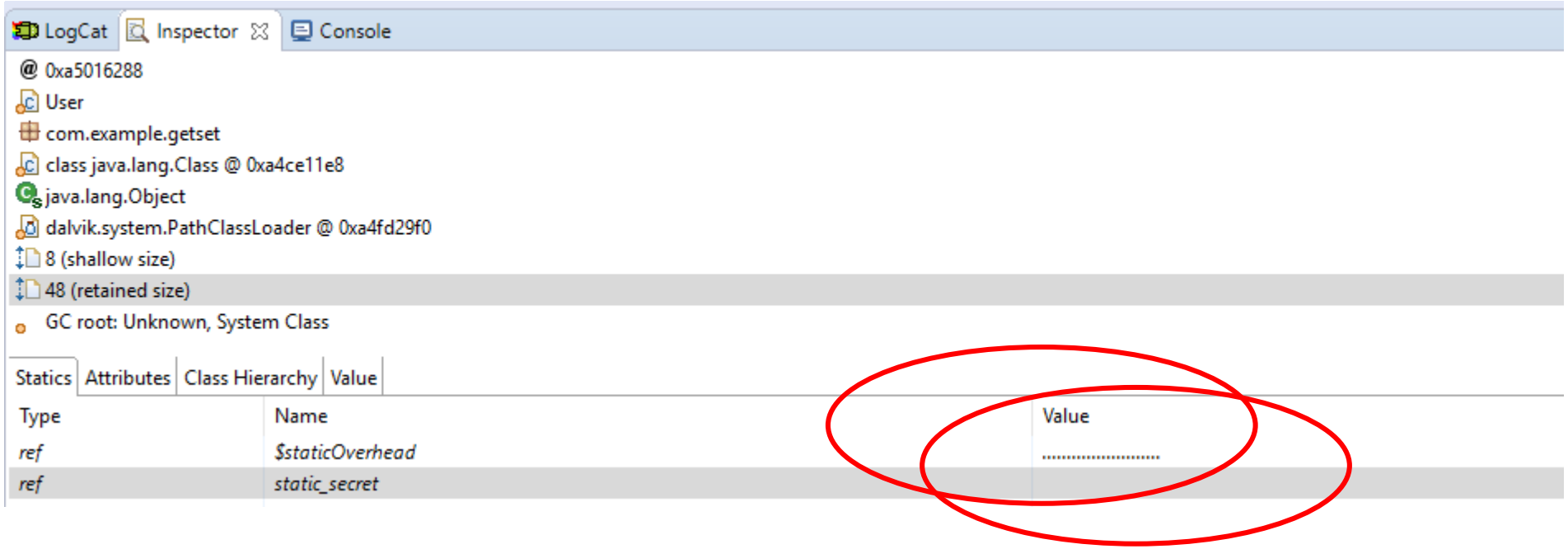



- Androsia determines last use of objects at a whole program level
  - A summary based inter-procedural data-flow analysis
- Androsia instruments bytecode to clear memory content of objects

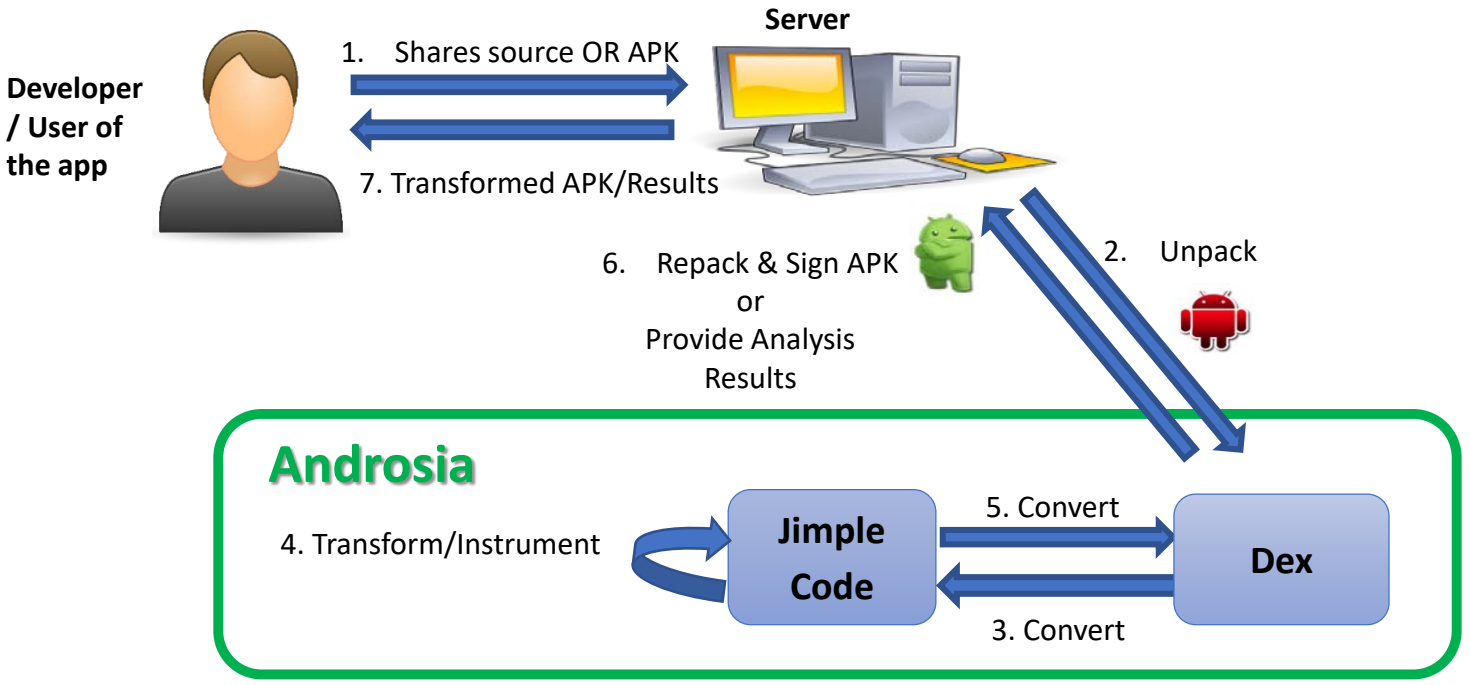
## Eclipse Memory Analyzer

**Heap Dump - Before Instrumentation**

**Heap Dump - After Instrumentation**



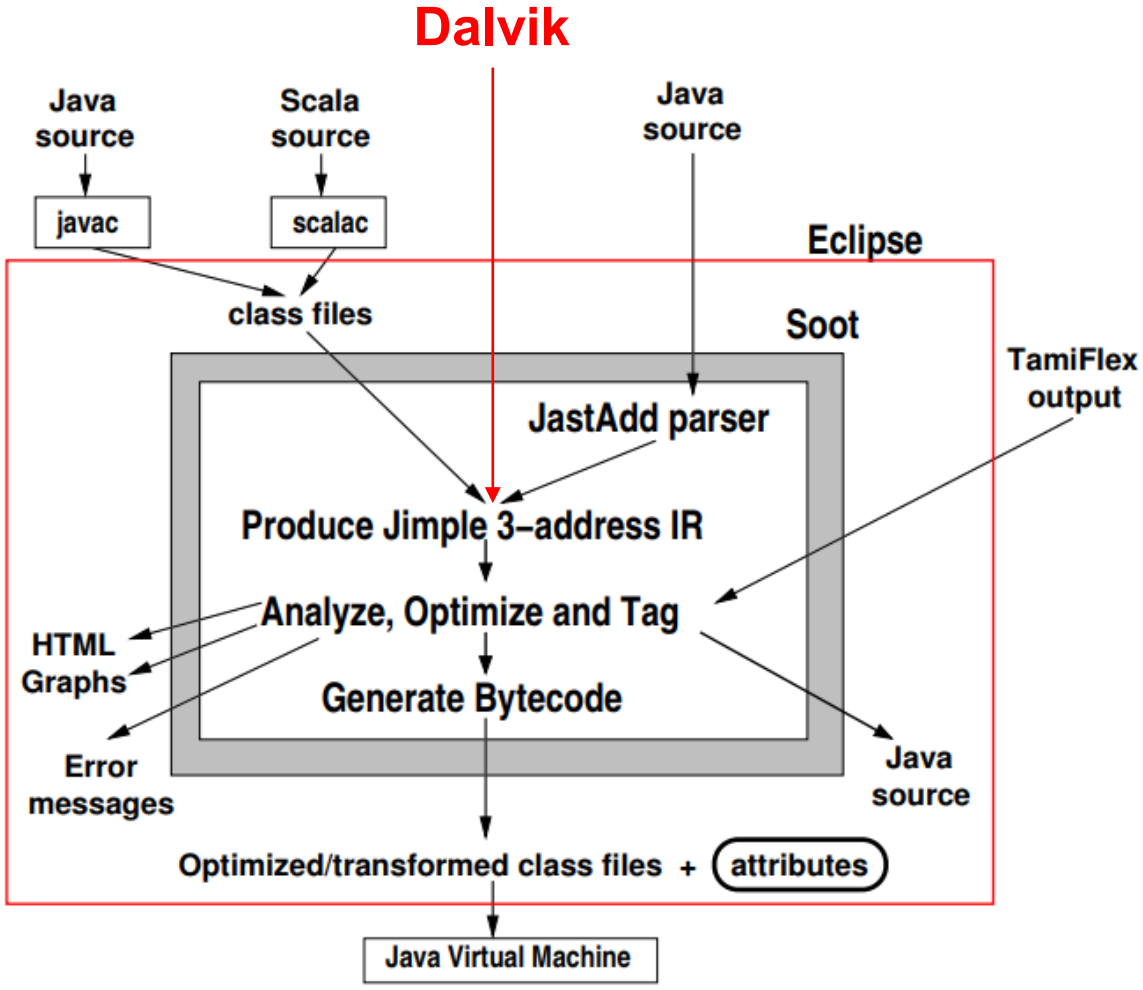
# Overview


Server
Developer / User of the app
1. Shares source OR APK
7. Transformed APK/Results
6. Repack & Sign APK or Provide Analysis Results
2. Unpack
Androsia
4. Transform/Instrument
Jimple Code
5. Convert
3. Convert
Dex

# Framework behind Androsia

# Static Code analysis using Soot

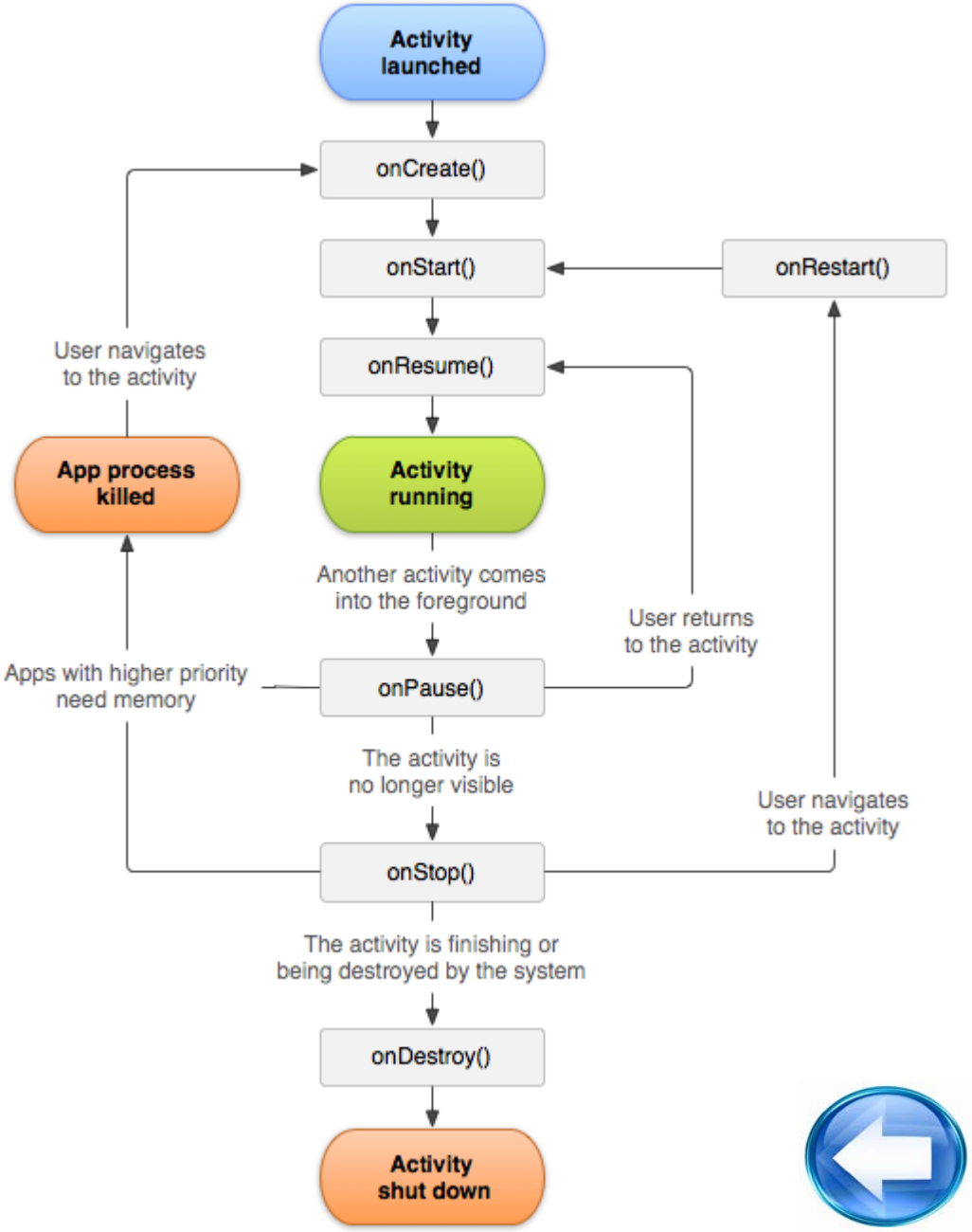
- Soot - framework for Java (bytecode), enables development of static analysis tools
  - Provides three-address code called Jimple
  - Supports implementing dataflow analyses:
    - Intra-procedural
    - Inter-procedural
- Soot was missing a Dalvik to Jimple transformation module
  - and then came Dexpler




**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/
Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot, https://arxiv.org/pdf/1205.3576.pdf

Img. ref: https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

# Demo

# SB Objects – In what scopes can they exist?

## Local variable

```
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

```
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()) {

        [z = y;]4

    } else{

        bar();5

    }]3
}}
```

Abbrev.
SB: StringBuilder

```
public static void bar(){
    System.out.println(StaticSB.x);
}
```

# Instance Field

```
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;

    }
}
```

# Demo - Static SB

```
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);
}
}
```

DEMO

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?

Data Liveness

- 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 $LV_{entry}$ & $LV_{exit}$ set for every statement
  - $LV_{entry}$ & $LV_{exit}$ → 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 *must* have a subsequent USE prior to next definition of that variable

Last Usage Point of a var ≅
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

Abbrev.
SB: StringBuilder

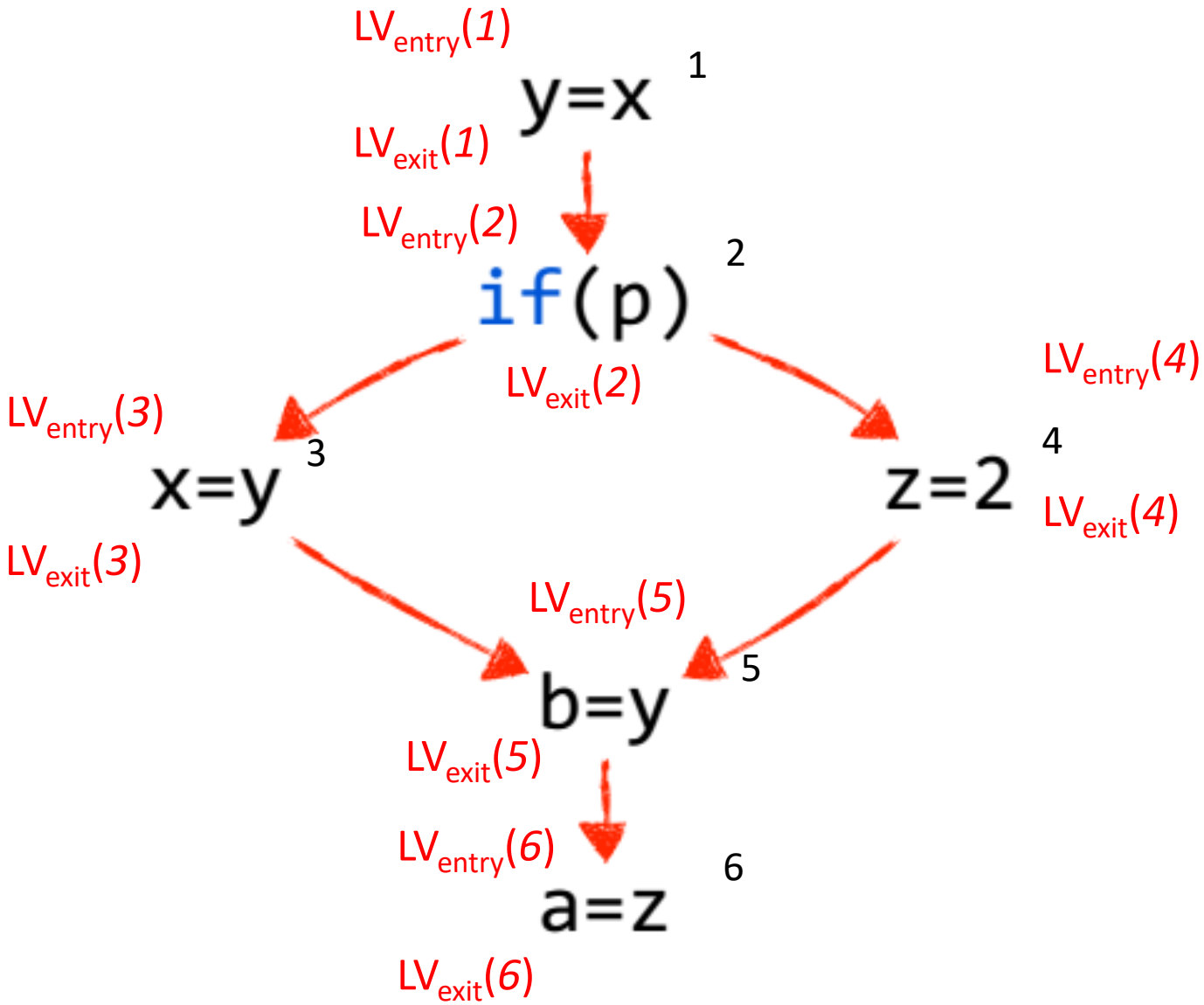
```
public void foo(){
SB x, y, z;

[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 (cntd.)

| l | *def(l)* | *use(l)* |
|---|---|---|
| 1 | { x } | ∅ |
| 2 | { y } | ∅ |
| 3 | { x } | ∅ |
| 4 | ∅ | { x, y } |
| 5 | { z } | { y } |
| 6 | { z } | { y } |
| 7 | { x } | { z } |

```
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
```

# LV Data Flow Direction



# 2. Compute $LV_{entry}(l)$ & $LV_{exit}(l)$

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

$$LV_{exit}(l) = \begin{cases} \emptyset & , \text{ if } l = b_{last} \\ \bigcup_{s \in succ[l]} \{LV_{entry}(s)\} & , \text{ otherwise} \end{cases}$$

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

# 3: Compute $LV_{entry}(l)$ & $LV_{exit}(l)$

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

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

| l | *def(l)* | *use(l)* |
|---|---|---|
| 1 | { x } | ∅ |
| 2 | { y } | ∅ |
| 3 | { x } | ∅ |
| 4 | ∅ | { x, y } |
| 5 | { z } | { y } |
| 6 | { z } | { y } |
| 7 | { x } | { z } |

| l | $LV_{entry}(l)$ | $LV_{exit}(l)$ |
|---|---|---|
| 1 | ∅ | ∅ |
| 2 | ∅ | { y } |
| 3 | { y } | { x, y } |
| 4 | { x, y } | { y } |
| 5 | { y } | { z } |
| 6 | { y } | { z } |
| 7 | { z } | ∅ |

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

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

LVentry(3) { y }
LVexit(3) { x, y }

LVentry(6) { y }
LVexit(6) { z }

LVentry(7) { z }
LVexit(7) ∅

# Summary is computed in reverse topological order

Summ(bar) = {sfr, ∅}

```
public void foo(){

A1
A2
A3 bar();

A4
A5

}
```

Summ(bar)

Summ(bar) = {sfr, B5}

```
public void bar(){
B1
B2
B3 baz();

B4
B5 sfr used

}
```

Summ(baz)

Summ(baz) = {sfr, C4}

```
public void baz(){
C1
C2
C3
C4 sfr used
C5
C6
}
```

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

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

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

$LV_{entry}(C4) = \{sfr, C4\}$

**Program level last use for "sfr" happens at:**

~~{ baz, (sfr,C4) }~~

{ bar, (sfr,B5) }

# Summarizing:

- 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 $LV_{entry}$ & $LV_{exit}$ set for every statement ✓
  - $LV_{entry}$ & $LV_{exit}$ → 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

# Demo - Instance Field SB

- 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

**DEMO**

# Work In Progress

- 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/

# References


1. Implementing an intra procedural data flow analysis in Soot, https://github.com/Sable/soot/wiki/Implementing-an-intra-procedural-data-flow-analysis-in-Soot

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

3. Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot, https://arxiv.org/pdf/1205.3576.pdf

4. Precise Interprocedural Dataflow Analysis via Graph Reachability, 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

# Three address code

```
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;
}
```