Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges And Solutions for Analyzing Android

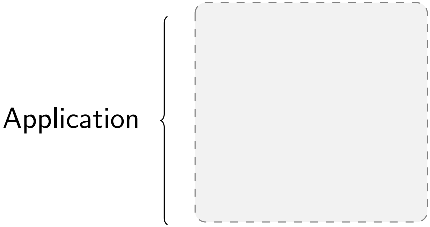
Alexandre Bartel

University of Luxembourg

September 8, 2014

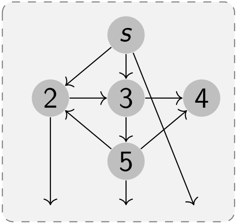
Supervisor: Yves Le Traon
Advisors: Jacques Klein & Martin Monperrus

Static Analysis of a Permission-Based Security System



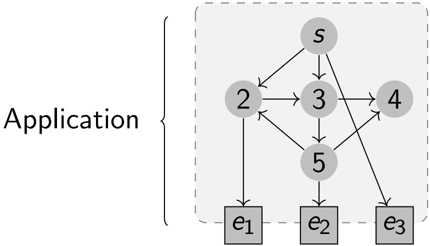
Static Analysis of a Permission-Based Security System

Application



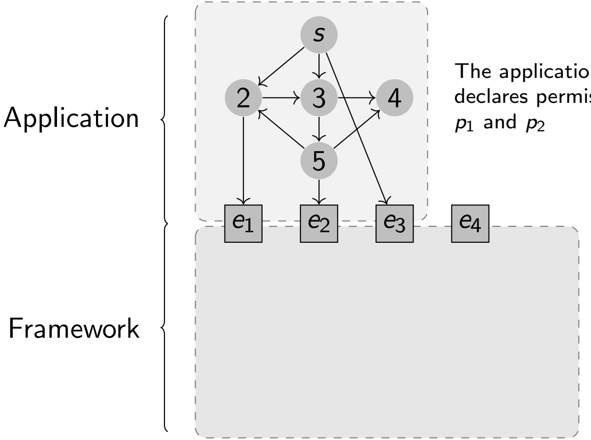
The application declares permissions p_1 and p_2

Static Analysis of a Permission-Based Security System



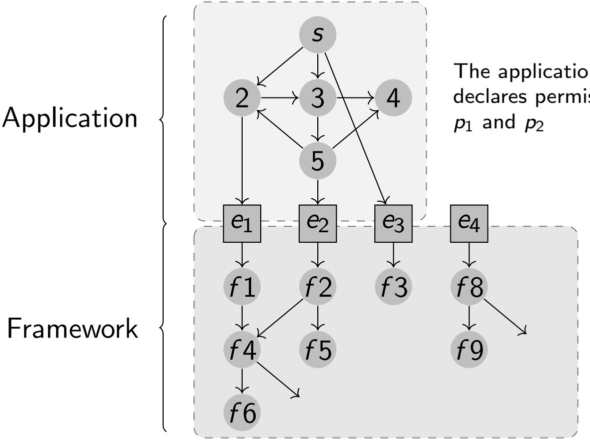
The application declares permissions p_1 and p_2

Static Analysis of a Permission-Based Security System



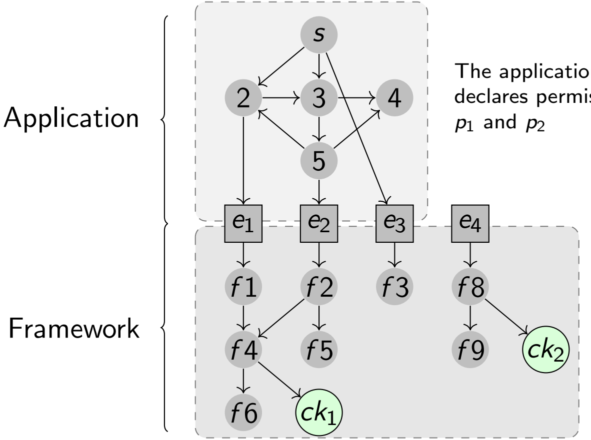
The application declares permissions p_1 and p_2

Static Analysis of a Permission-Based Security System



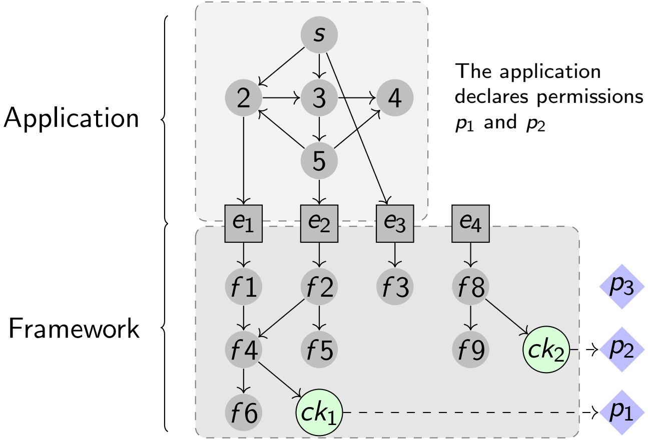
The application declares permissions p_1 and p_2

Static Analysis of a Permission-Based Security System



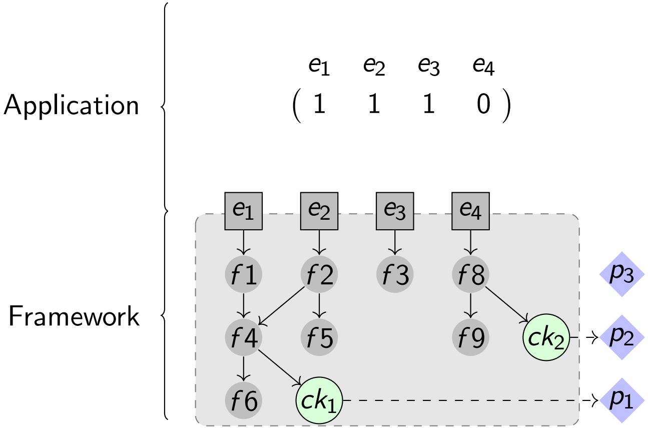
The application declares permissions p_1 and p_2

Static Analysis of a Permission-Based Security System



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

$$\text{Application}\left\{\quad \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \\ \end{array} \atop \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}\right.$$

$$\text{Framework}\left\{\quad \begin{array}{c|c} & \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{array}{c} e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}\right.$$

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

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

This step is done *for every application*.

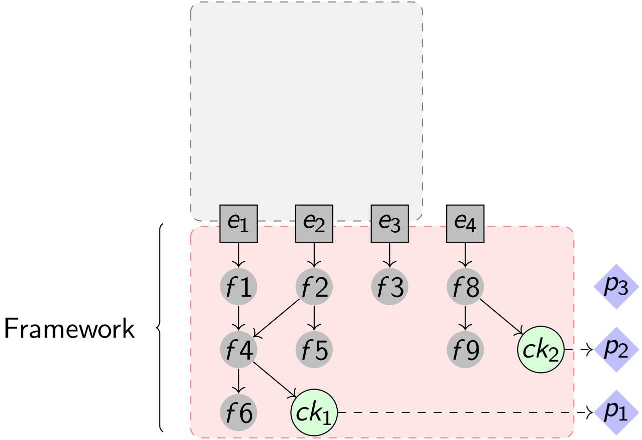
Methodology to Compute Permission Set (Step 3/3)

Step 3: Infer Permission Set of the Application

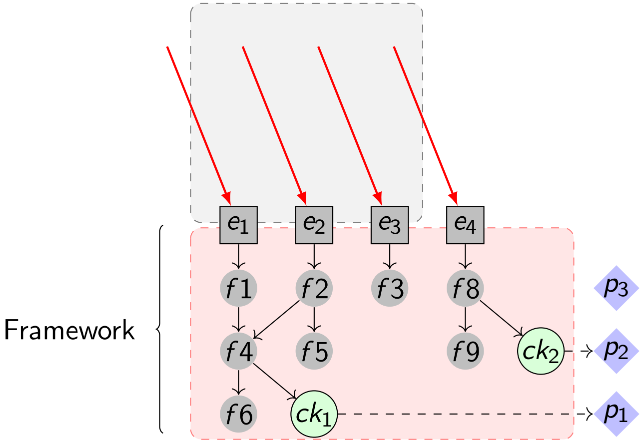
$$
\begin{aligned}
IP_{app} &= \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \\
IP_{app} &= \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}
\end{aligned}
$$

This step is done *for every application*.

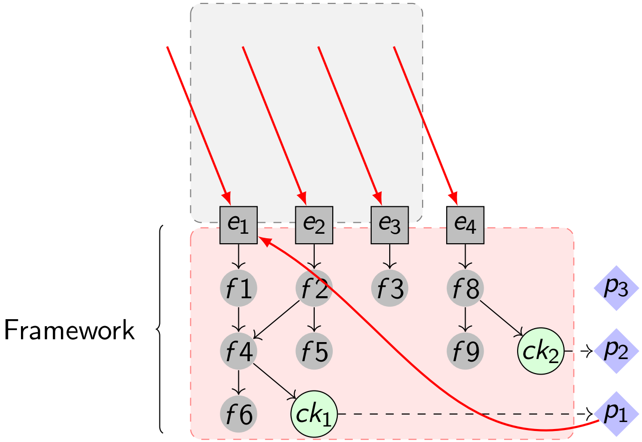
Android Framework Call Graph Construction



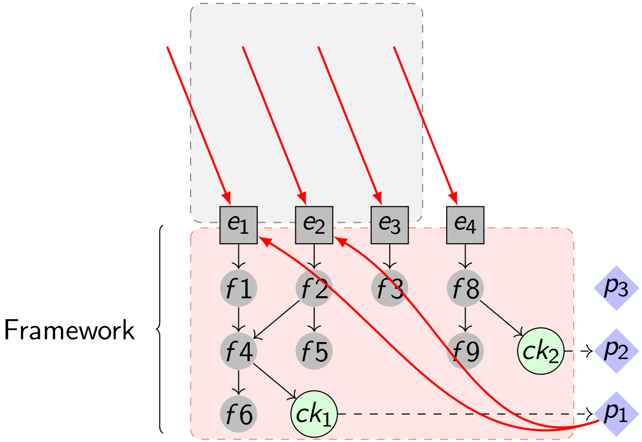
Android Framework Call Graph Construction



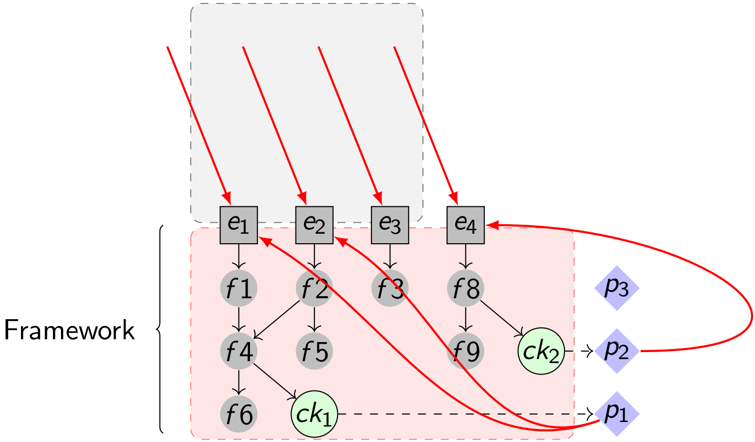
Android Framework Call Graph Construction



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?

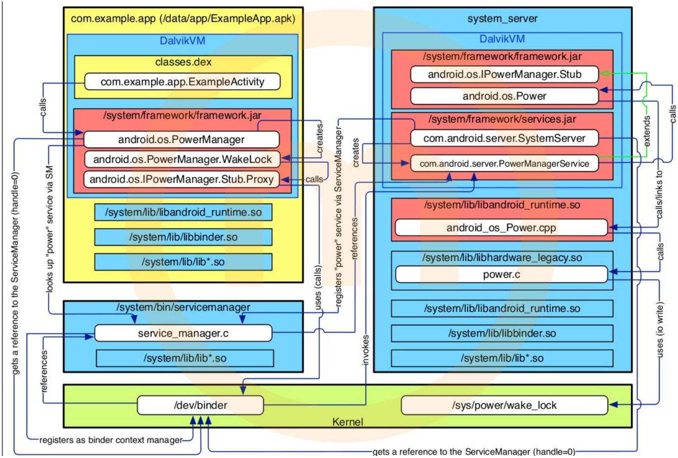
CHA Essential (1/4)

- 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%)

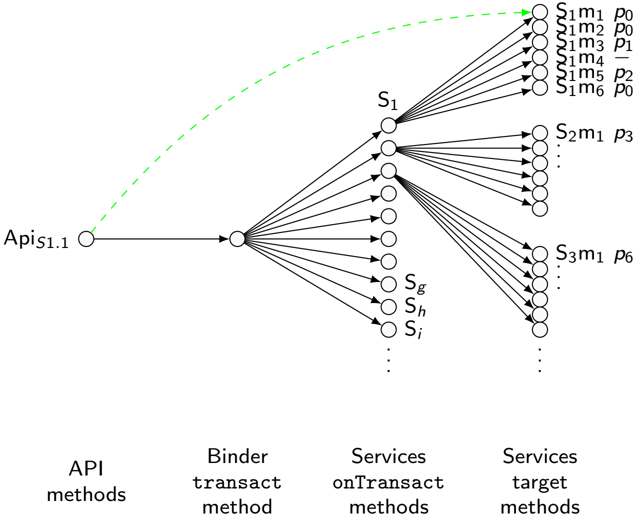
- Why explosion of permission set size?
  - Call graph goes through binder code

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



(source: Gargenta, 2012)

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



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

Application Code

Service Call

```
r = getSystemService()
p = r.getPassword()
```

Account System Service

```
getPassword() {
 checkPermission();
 return password;
}
```

Binder (Linux module)

CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

1 r = getSystemService()

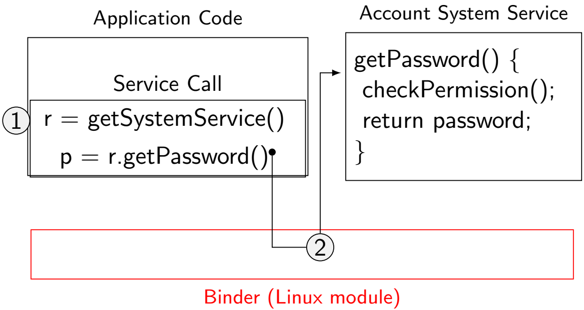
p = r.getPassword()

Account System Service

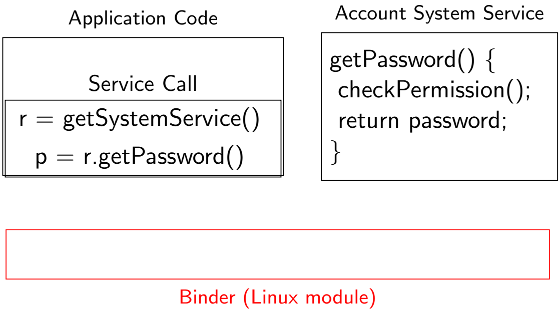
```
getPassword() {
 checkPermission();
 return password;
}
```

Binder (Linux module)

CHA Intelligent (2/4): Handling Binder



CHA Intelligent (2/4): Handling Binder



CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

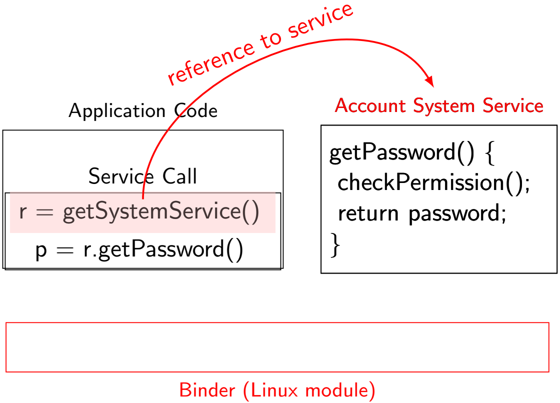
```
r = getSystemService()
p = r.getPassword()
```

Account System Service

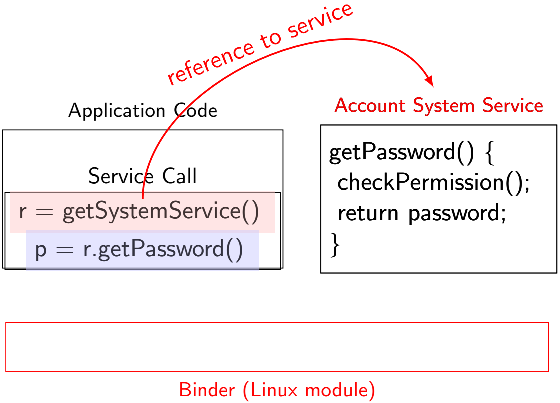
```
getPassword() {
 checkPermission();
 return password;
}
```

Binder (Linux module)

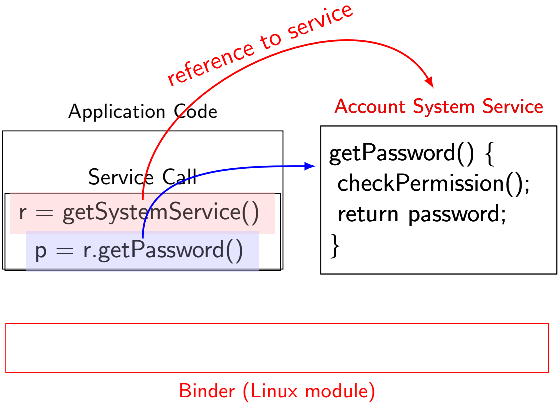
CHA Intelligent (2/4): Handling Binder



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)

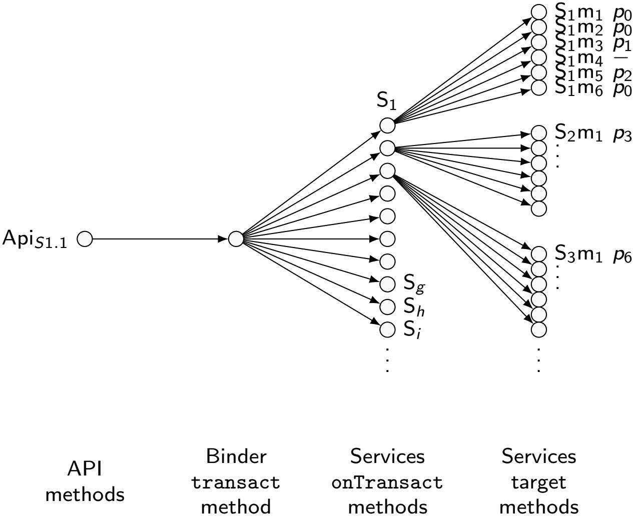
Spark Intelligent (4/4)

- Generates entry point wrappers

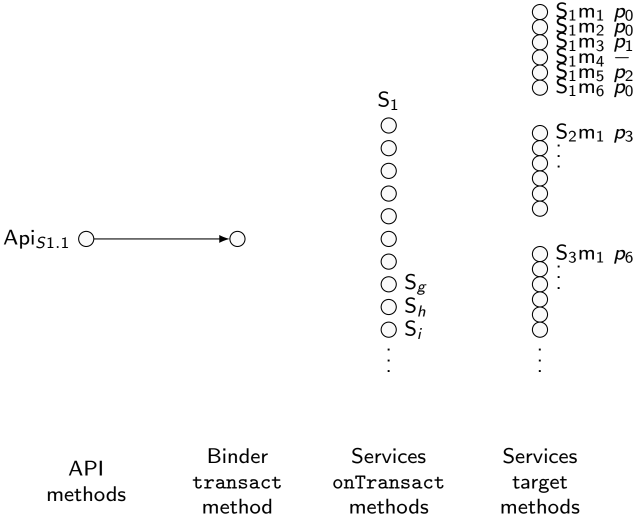
Spark Intelligent (4/4)

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

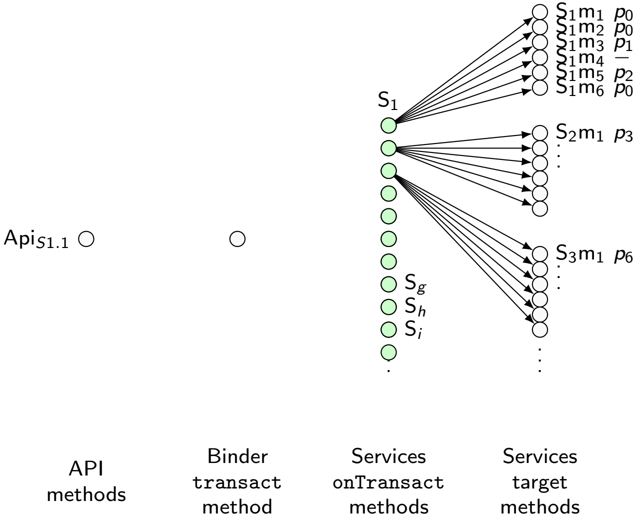
Spark Intelligent (4/4)



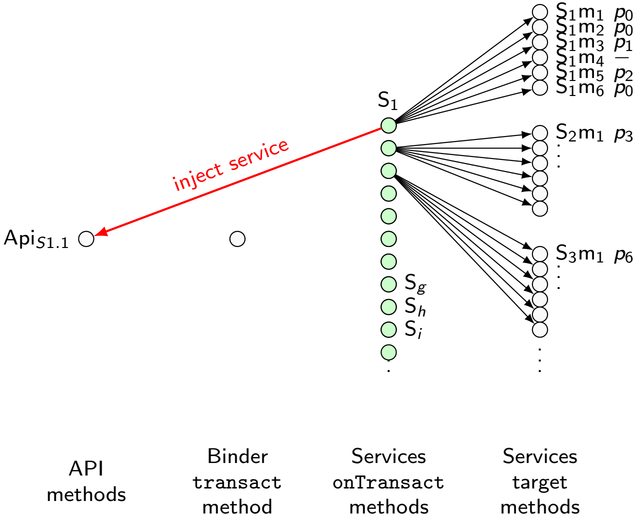
Spark Intelligent (4/4)



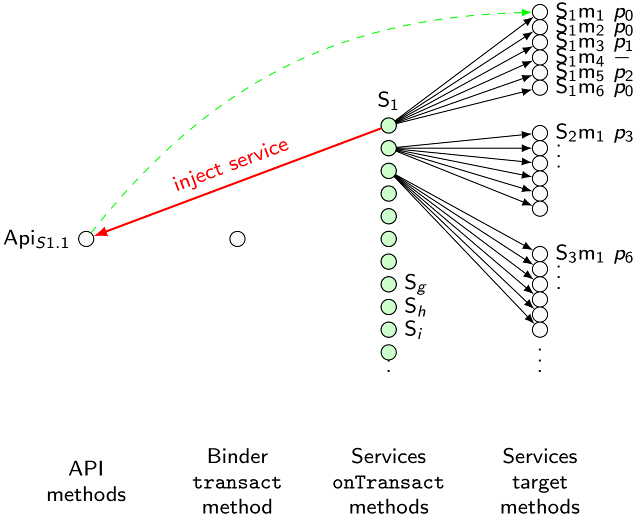
Spark Intelligent (4/4)



Spark Intelligent (4/4)



Spark Intelligent (4/4)



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

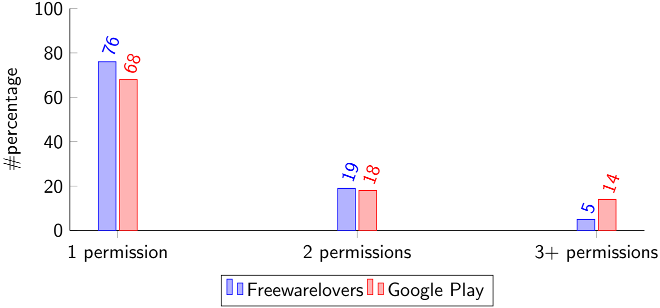
- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications:

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.