Smart Phone Technology Security

The Case of Android: Systems, Attacks, Defenses

Haipeng Cai
School of Electrical Engineering and Computer science
Washington State University

Email: haipeng.cai@wsu.edu
Webpage: http://eecs.wsu.edu/~hcai

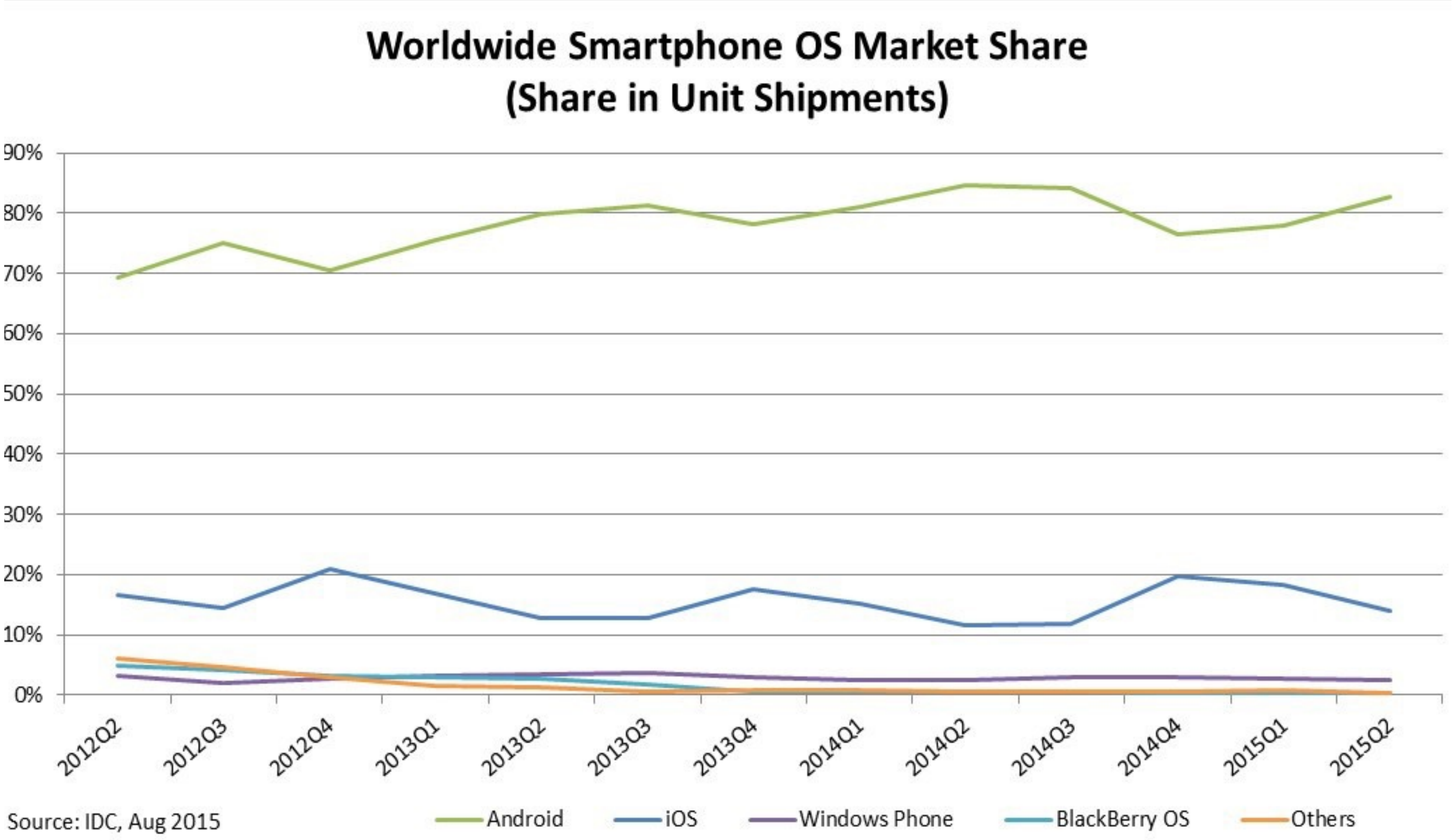
NORTHWEST VIRTUAL INSTITUTE FOR

CYBERSECURITY EDUCATION AND RESEARCH

Outline

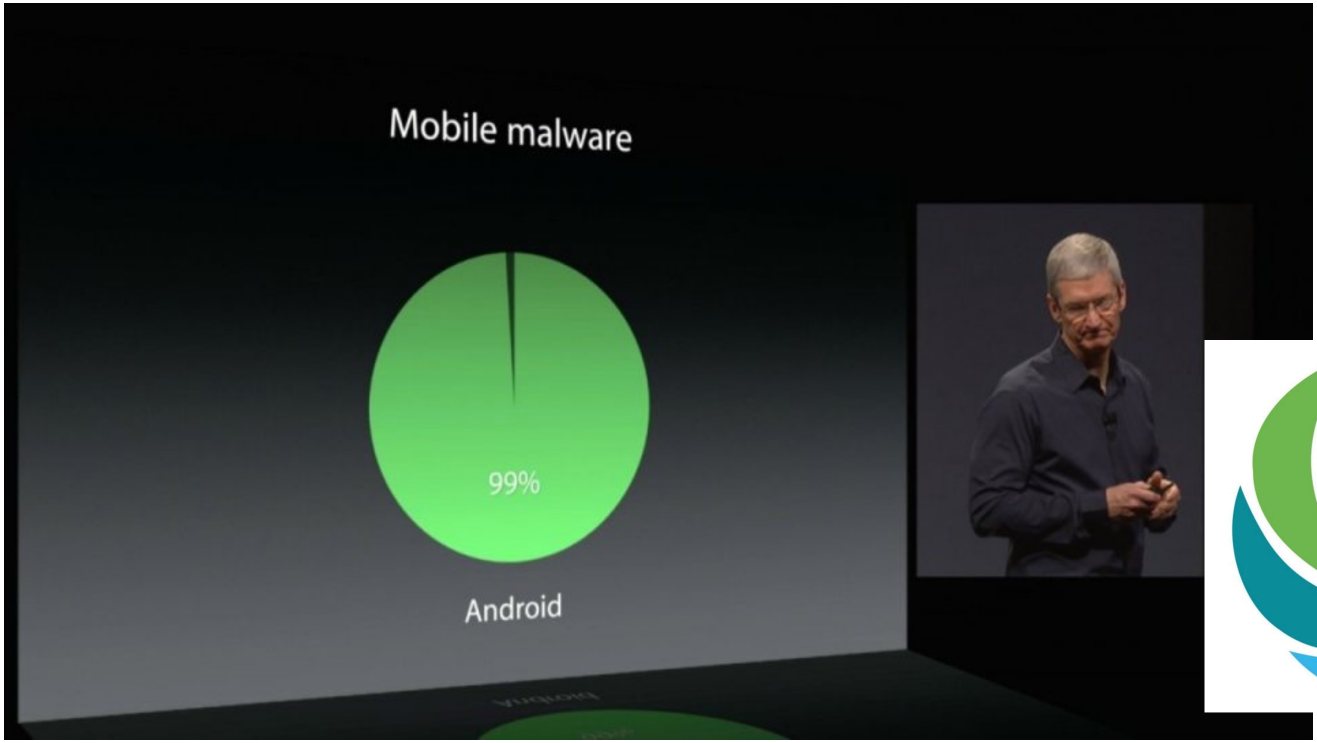
- Background
  - Mobile software and Android
- System/apps
  - Android system
  - Android apps
  - Android security mechanisms
- Attacks
  - Security attacks on systems/apps
- Defenses
  - Current defenses against the attacks
- Summary
  - Takeaways

Mobile market trends



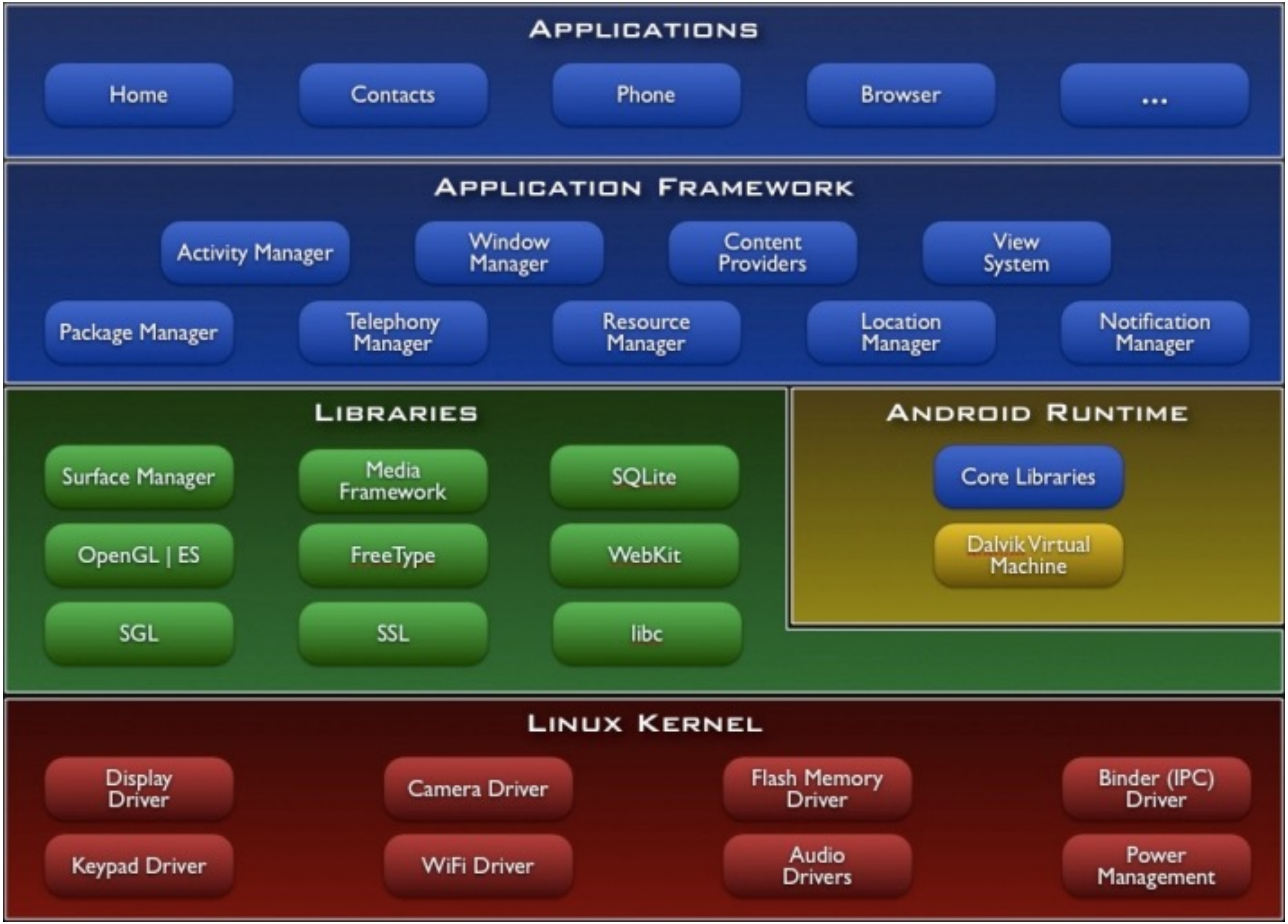
Source: IDC, Aug 2015

Android as the target



Android dominates mobile computing platforms …

Android dominates even more in malware market…


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Android Platform

- Linux kernel, browser, SQL-lite database
- Software for secure network communication
  - Open SSL, Bouncy Castle crypto API and Java library
- C language infrastructure
- Java platform for running applications
  - Dalvik bytecode, virtual machine

Android Apps

Each Android app contains one or more components of the following types:

- Activity
  - Portions of the application's user interface
    - Login window, registration interface, etc.
- Service
  - Performs background processing
    - Download a file, play music, etc.
- Broadcast Receiver
  - Handlers for global messages
    - Boot completed, power disconnected, etc.
- Content Provider
  - Manages access to structured data
    - User calendar, contacts, etc.

Each component runs as a separate thread in the OS

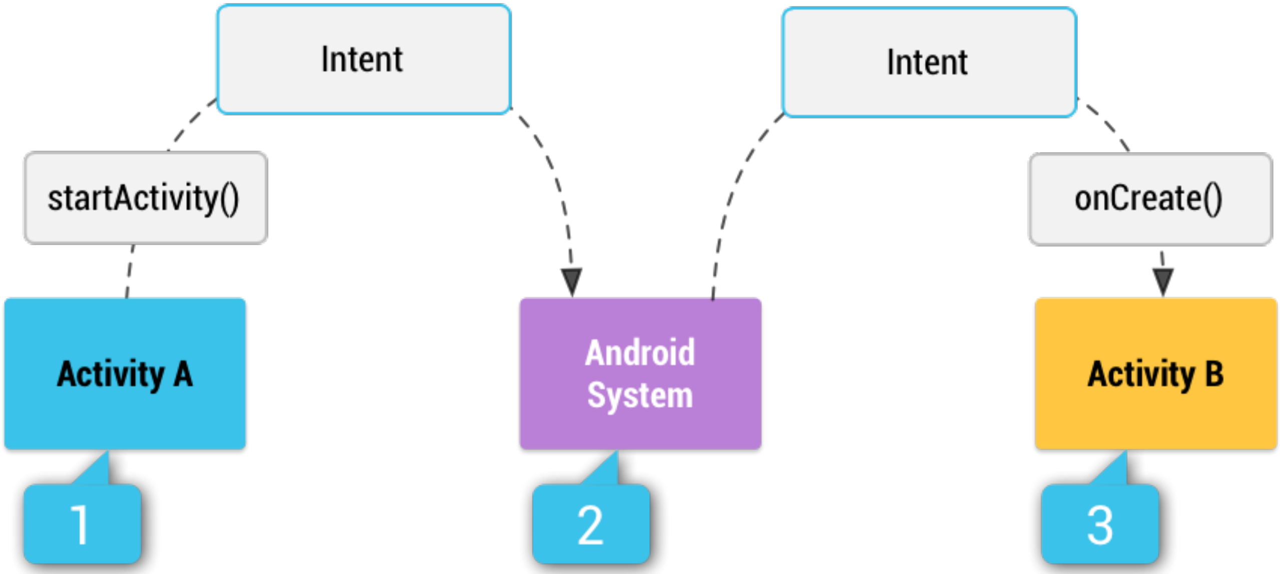
Application Structure

Inter-Component Communication (ICC)

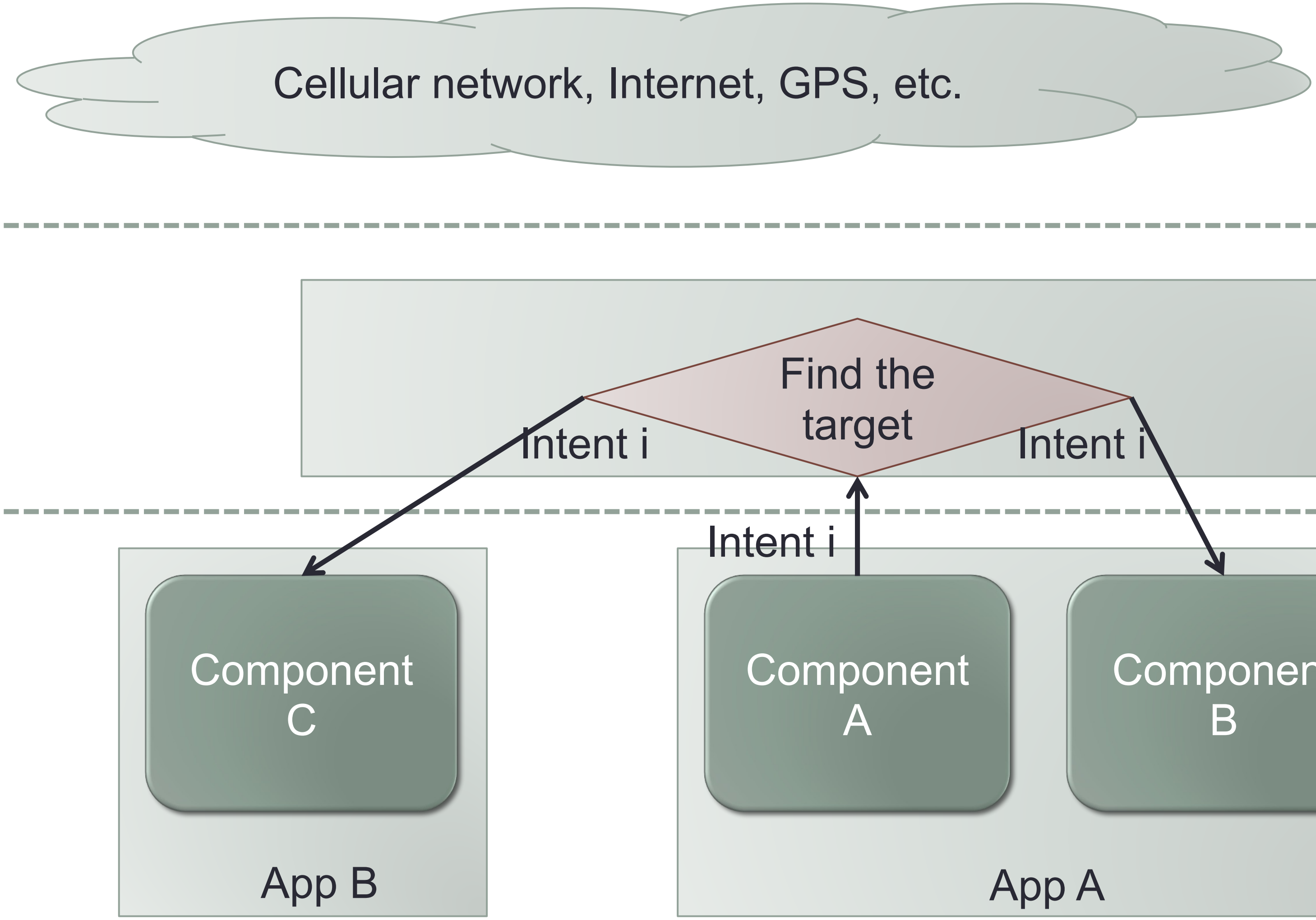
- Apps need to communicate with each other and the system
  - A restaurant recommender app may need to launch a map app to show a restaurant's location on map
  - An email app may need to launch a PDF viewer to open an attachment
  - A messenger app may need to receive text messages sent to the phone
- Component interaction
  - **Intent** - is the primary mechanism for component interaction, which is simply a message object containing a destination component address and data

Intents - Explicit vs. Implicit

- Explicit Intents specify a component to start.
- Implicit Intents give a general action to perform.

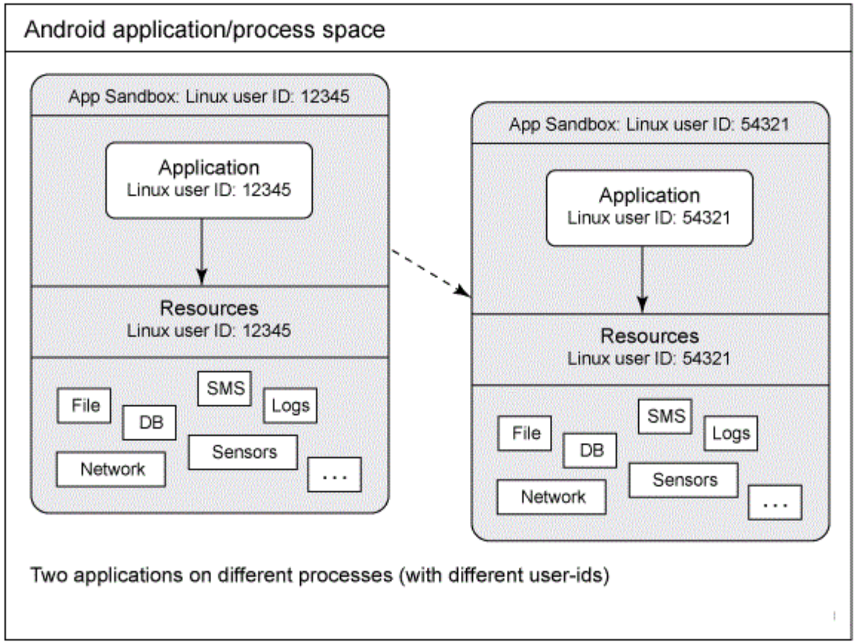


How Android app works

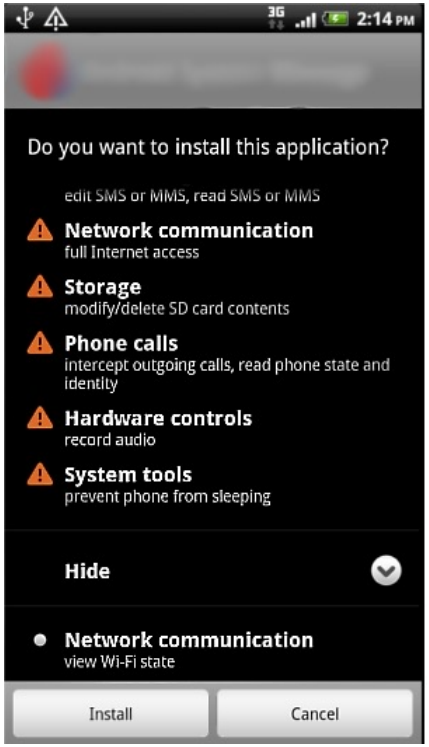


Security mechanisms

- Two main Android security mechanisms
  - Sandbox
  - Permission



Two applications on different processes (with different user-ids)



Do you want to install this application?

edit SMS or MMS, read SMS or MMS

⚠ **Network communication**
full Internet access

⚠ **Storage**
modify/delete SD card contents

⚠ **Phone calls**
intercept outgoing calls, read phone state and identity

⚠ **Hardware controls**
record audio

⚠ **System tools**
prevent phone from sleeping

Hide

● **Network communication**
view Wi-Fi state

Application sandbox

- Every app runs as a separate user
  - Underlying Unix OS provides system-level isolation

- Each application runs with its UID in its own Dalvik virtual machine
  - Provides CPU protection, memory protection
  - Authenticated communication protection using Unix domain sockets
  - Only ping, zygote (spawn another process) run as root

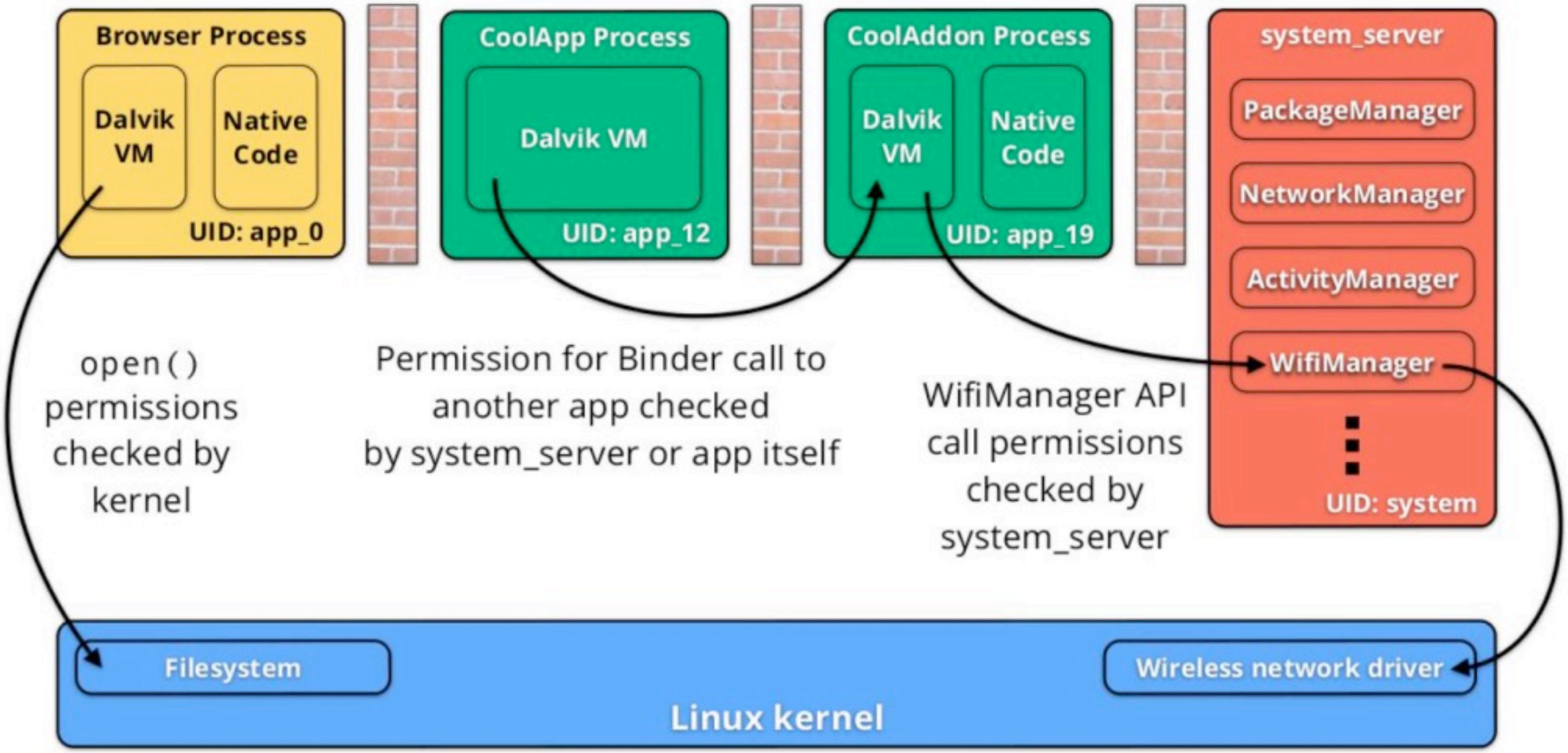
Android permissions

- Applications announce permission requirement
  - Create a whitelist model – user grants access
    - Don't interrupt user – all questions asked as install time
  - Inter-component communication reference monitor checks permissions

- Example of permissions provided by Android
  - "android.permission.INTERNET"
  - "android.permission.READ_EXTERNAL_STORAGE
  - "android.permission.SEND_SMS"
  - "android.permission.BLUETOOTH"

- Also possible to define custom permissions

Security mechanisms



open() permissions checked by kernel

Permission for Binder call to another app checked by system_server or app itself

WifiManager API call permissions checked by system_server

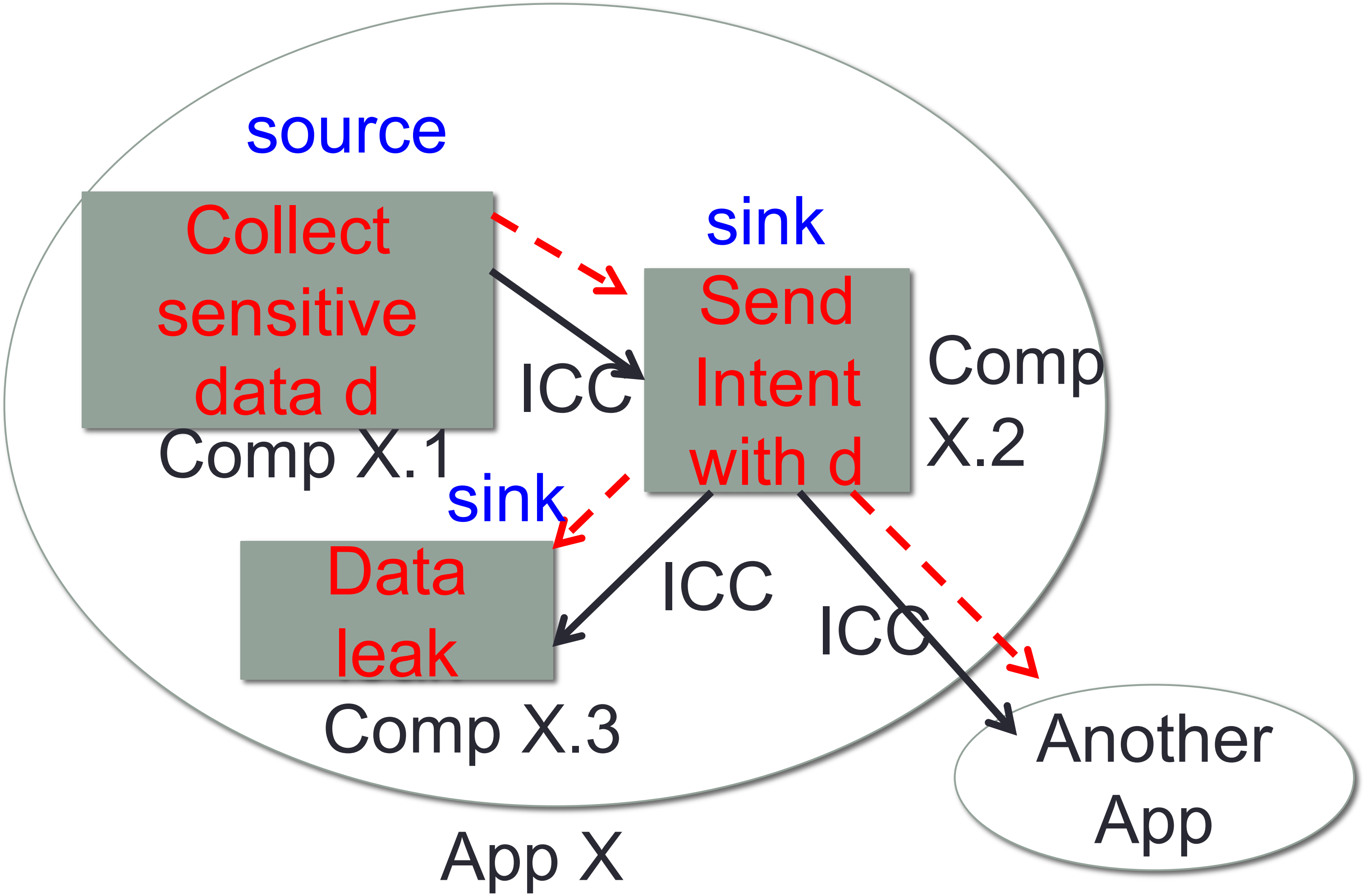
Security mechanisms

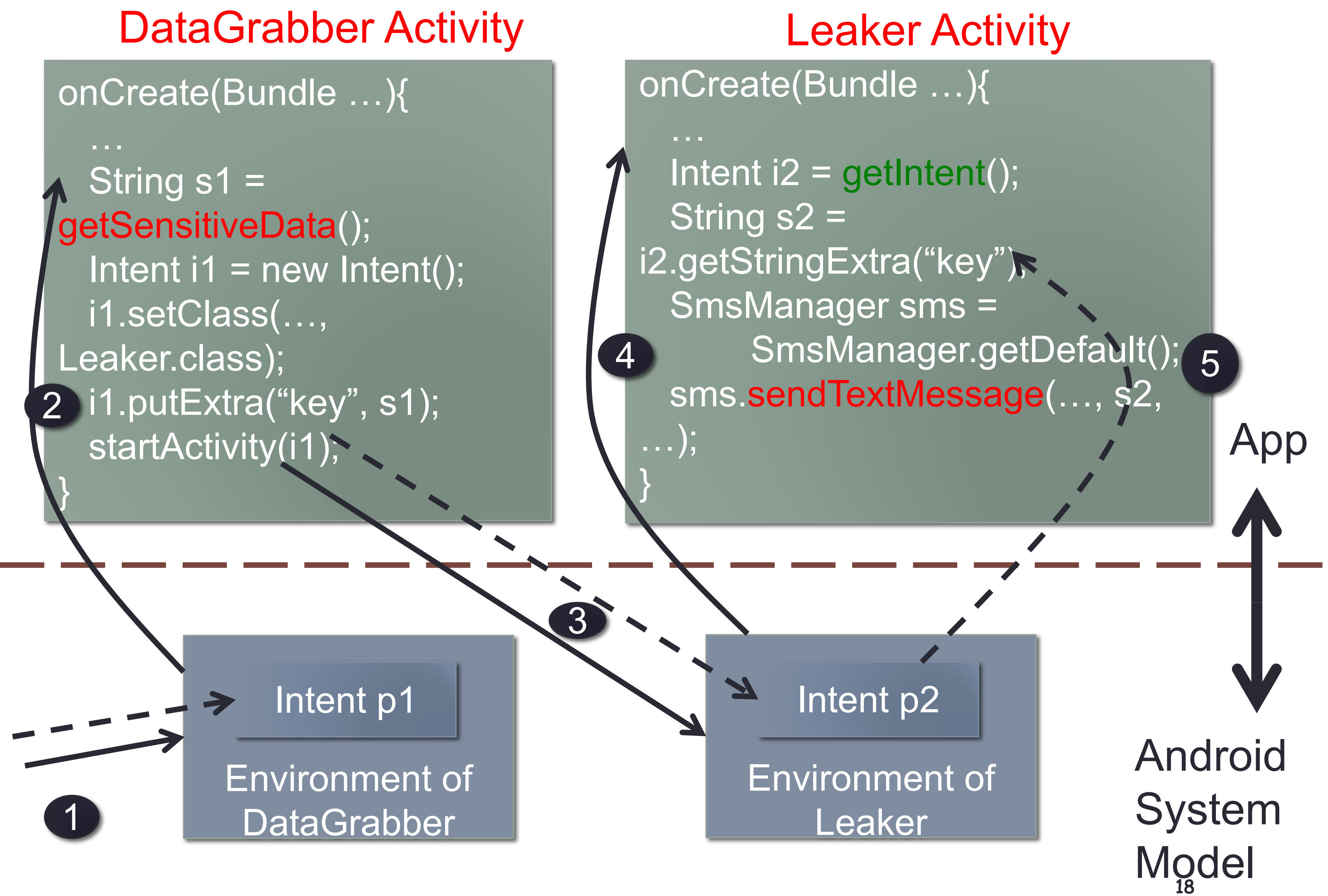
- Specifying protection domain via permission labels
- Mandatory (permission-based) policy enforcement
- No information flow guarantees
- Private versus public components
- No access permission specified = all access!
- Intent access control against broadcasting privacy leaks
- (Sensitive) API protection

General Security Vulnerabilities / Attacks

- Flaws in Android OS itself
- Flaws in phone software/firmware
- Conventional browser based virus
- Vulnerabilities within downloaded apps
- Unconventional attacks (injecting code into accelerometers i.e.)
- New classes of vulnerabilities
  - E.g.: Web advertiser gets to inject arbitrary code into mobile apps running on your phone!%#$!
- Evolving defenses

Data leakage





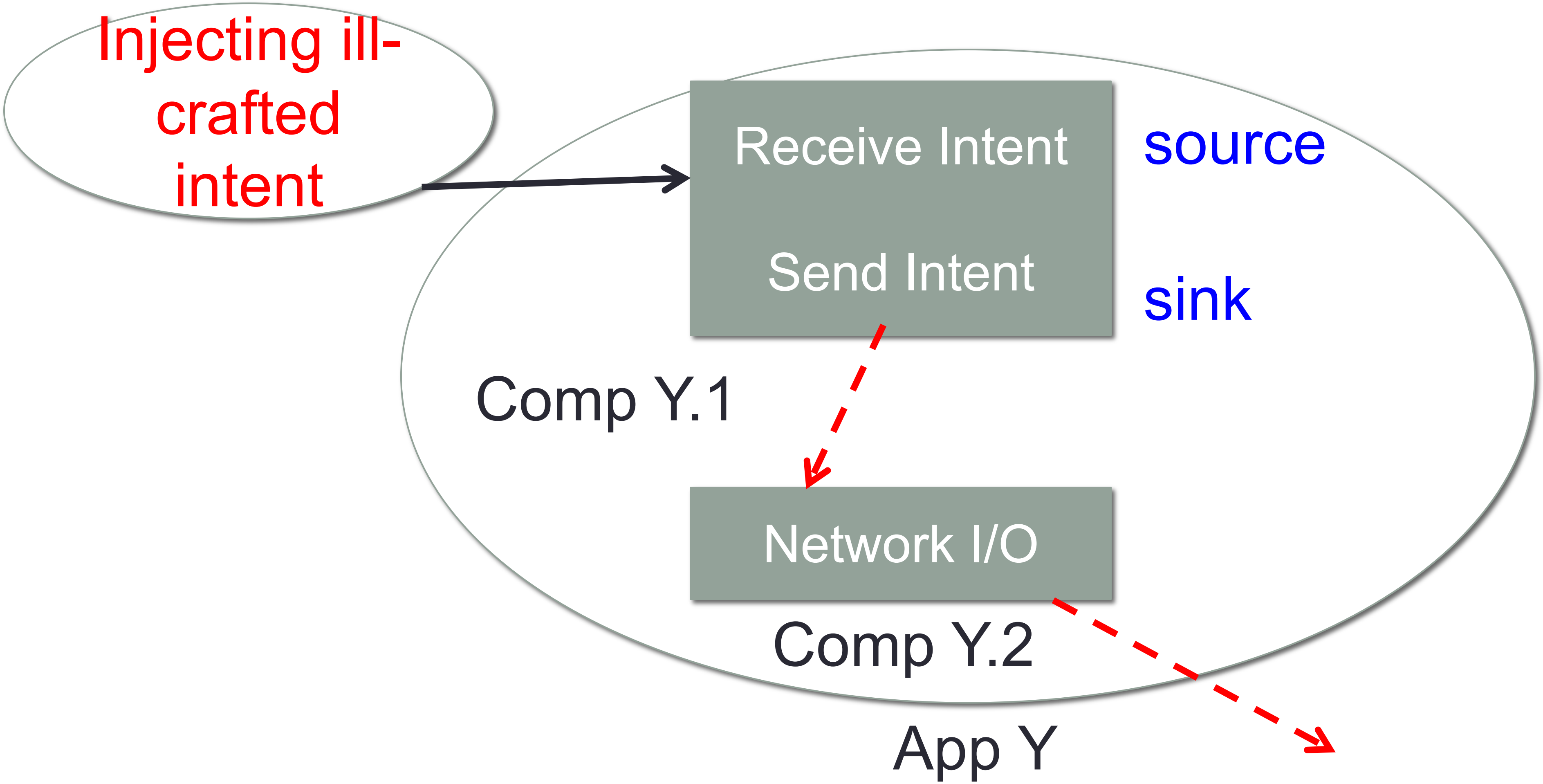
DataGrabber Activity

```
onCreate(Bundle …){
   …
   String s1 =
getSensitiveData();
   Intent i1 = new Intent();
   i1.setClass(…,
Leaker.class);
   i1.putExtra(“key”, s1);
   startActivity(i1);
}
```

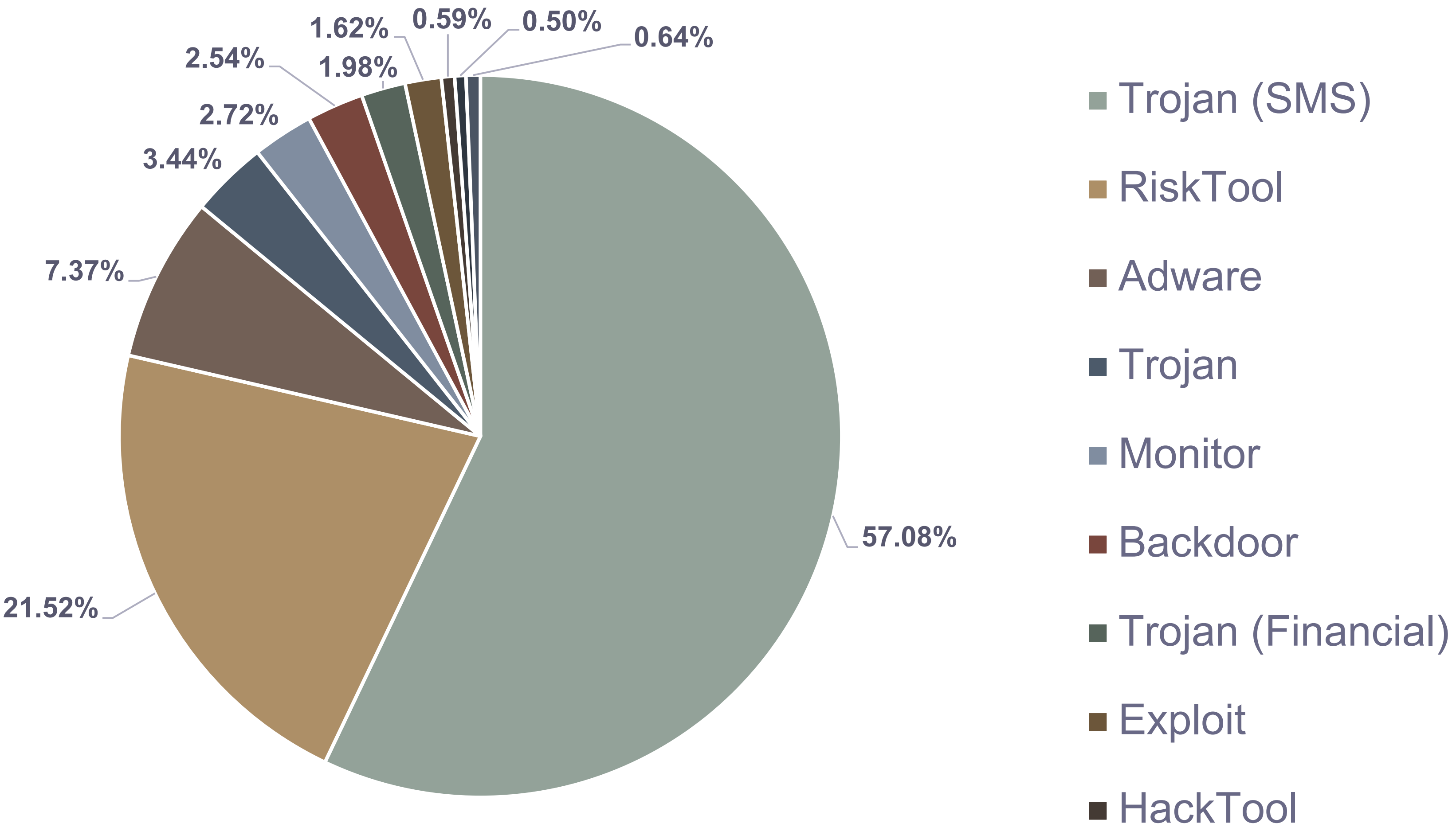
Leaker Activity

```
onCreate(Bundle …){
   …
   Intent i2 = getIntent();
   String s2 =
i2.getStringExtra(“key”);
   SmsManager sms =
          SmsManager.getDefault();
   sms.sendTextMessage(…, s2,
…);
}
```

Data injection



Malware Types



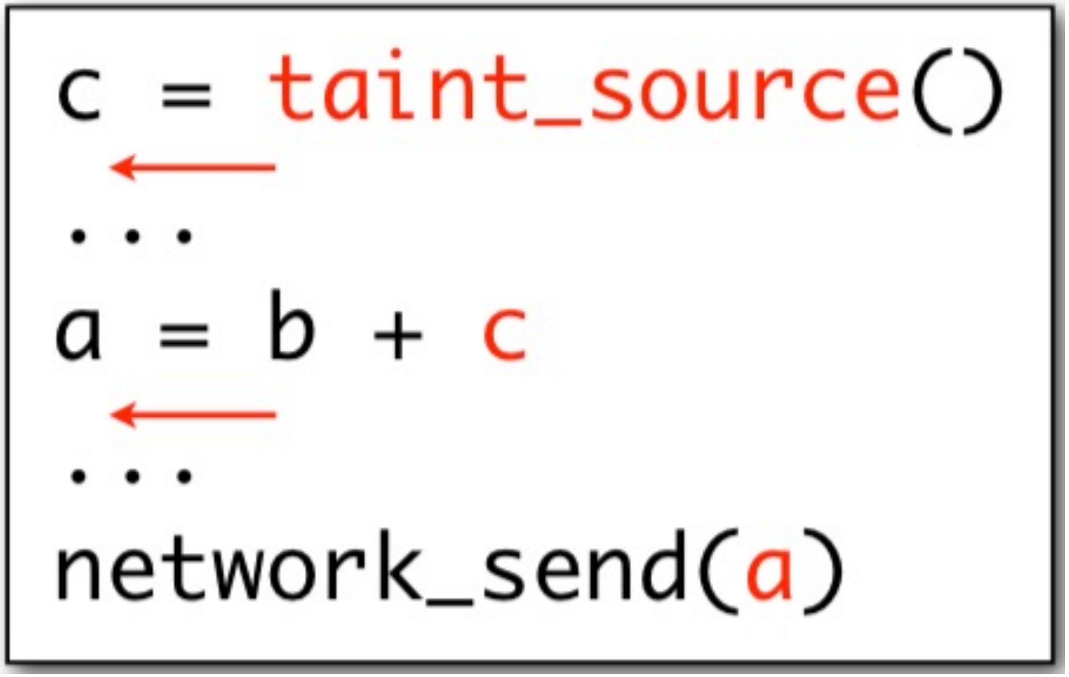
General Security Defenses

- Open source: public review, no obscurity
- Secure drivers, media codecs, new security features
- Strict access (e.g., permission) control
- Securing information flow (e.g., taint analysis)
- Memory protection (against overflow, ASLR)
- Malware countermeasures

Securing information flow (taint analysis)

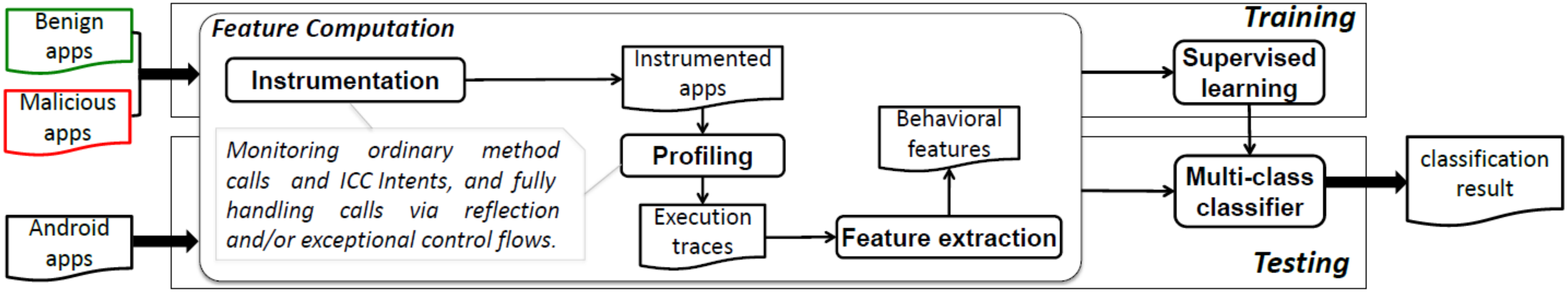
- DTA is a technique that tracks information dependencies from an origin

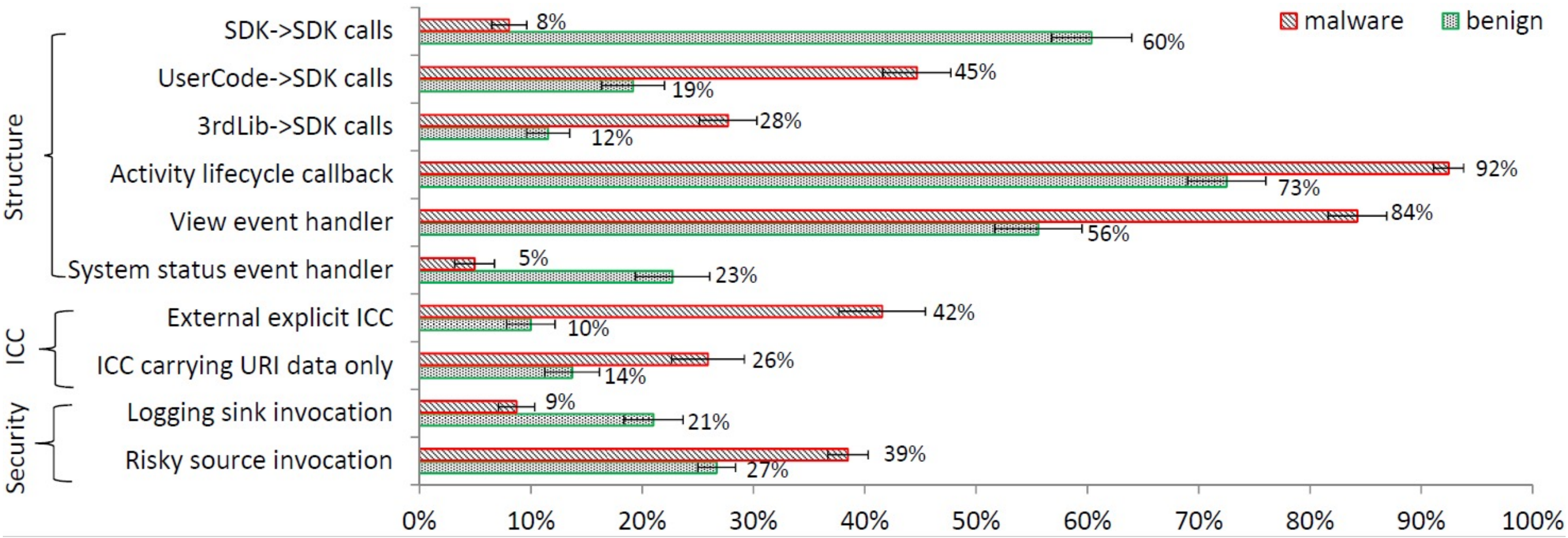
- High-level:
  - Taint source
  - Taint propagation
  - Taint sink



```
c = taint_source()
...
a = b + c
...
network_send(a)
```

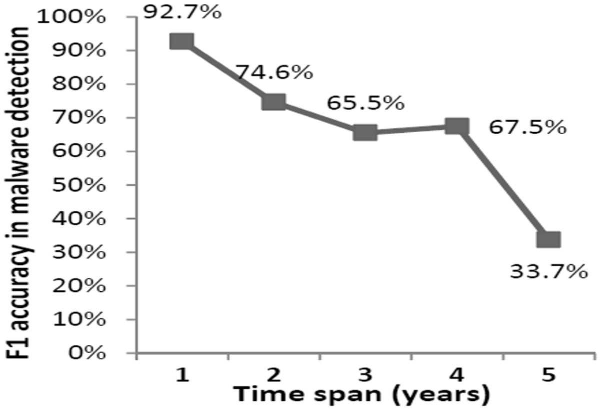
Malware detection (ML-based)





Sustainability - a new quality metric

Android evolution causing malware detection deterioration



- **Sustainability**

the accuracy of a classifier **trained** on apps of year *x* and **tested** against apps of year *y, y>=x*

- Reusability

the accuracy of a classifier **trained** on apps of year *x* and **tested** against apps of year *y, y == x*

*Accounting for how the classifier sustains **with** retraining*

- Stability

- the accuracy of a classifier **trained** on apps of year *x* and **tested** against apps of year *y, y > x*
- *y - x*

*Accounting for how the classifier sustains **without** retraining or other model updates*

DroidSpan - a detector based on SAD profiles

App evolution characterization

⇓

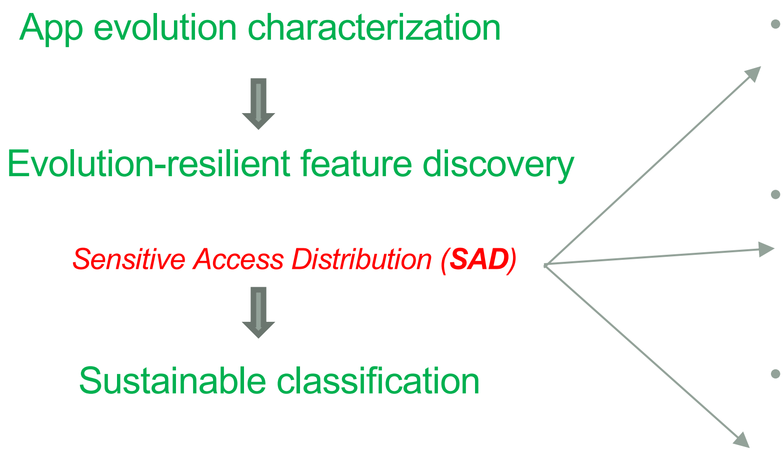
Evolution-resilient feature discovery

*Sensitive Access Distribution (**SAD**)*

⇓

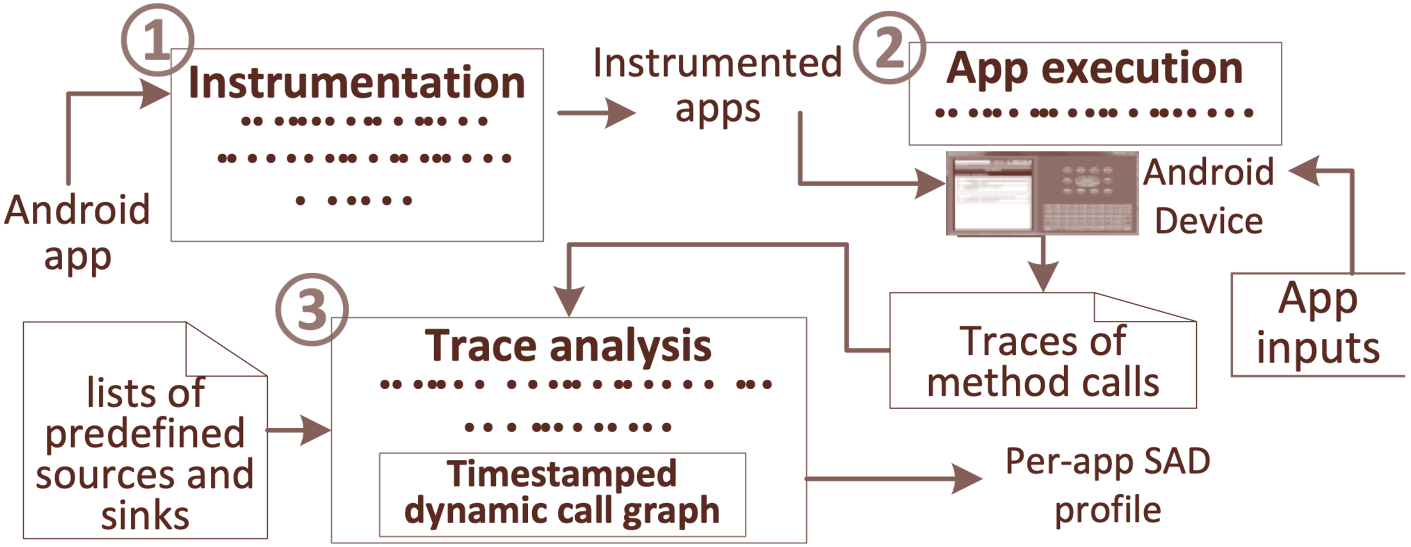
Sustainable classification

DroidSpan - a detector based on SAD profiles


App evolution characterization
Evolution-resilient feature discovery
*Sensitive Access Distribution (**SAD**)*
Sustainable classification

- Extent of sensitive access
  - *E.g., percentage of total source/sink callsites and call instances*
- Categorization sensitive data and operations accessed
  - *E.g., percentage of source/sink callsites retrieving network info*
- Vulnerable method-level control flows
  - *E.g., percentage of call instances to sources accessing Account data that reach at least a sink*

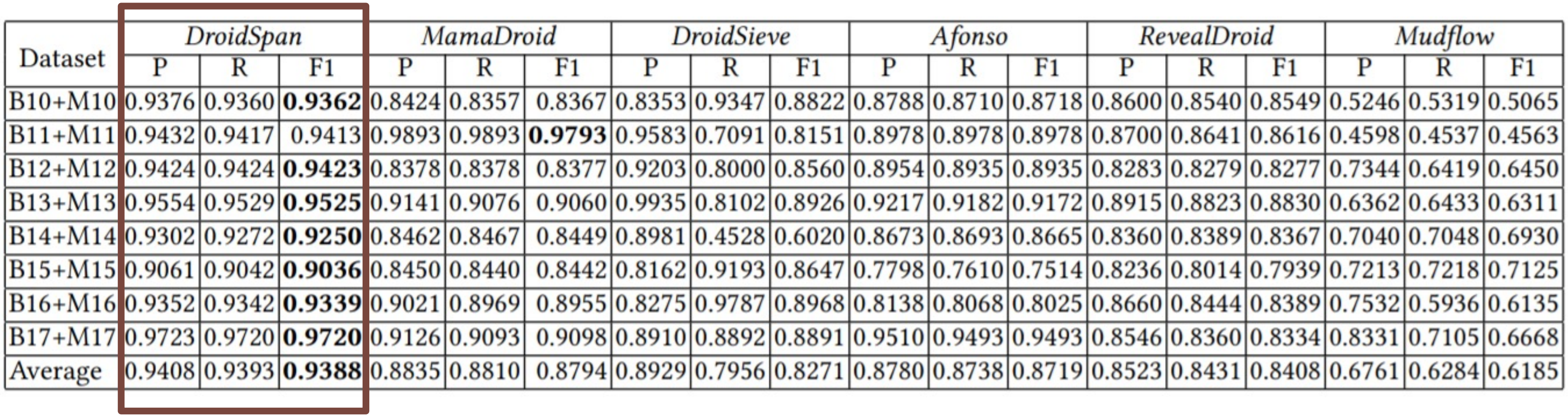
DroidSpan - a detector based on SAD profiles



Constructing the SAD profile of a given Android app

Results - reusability

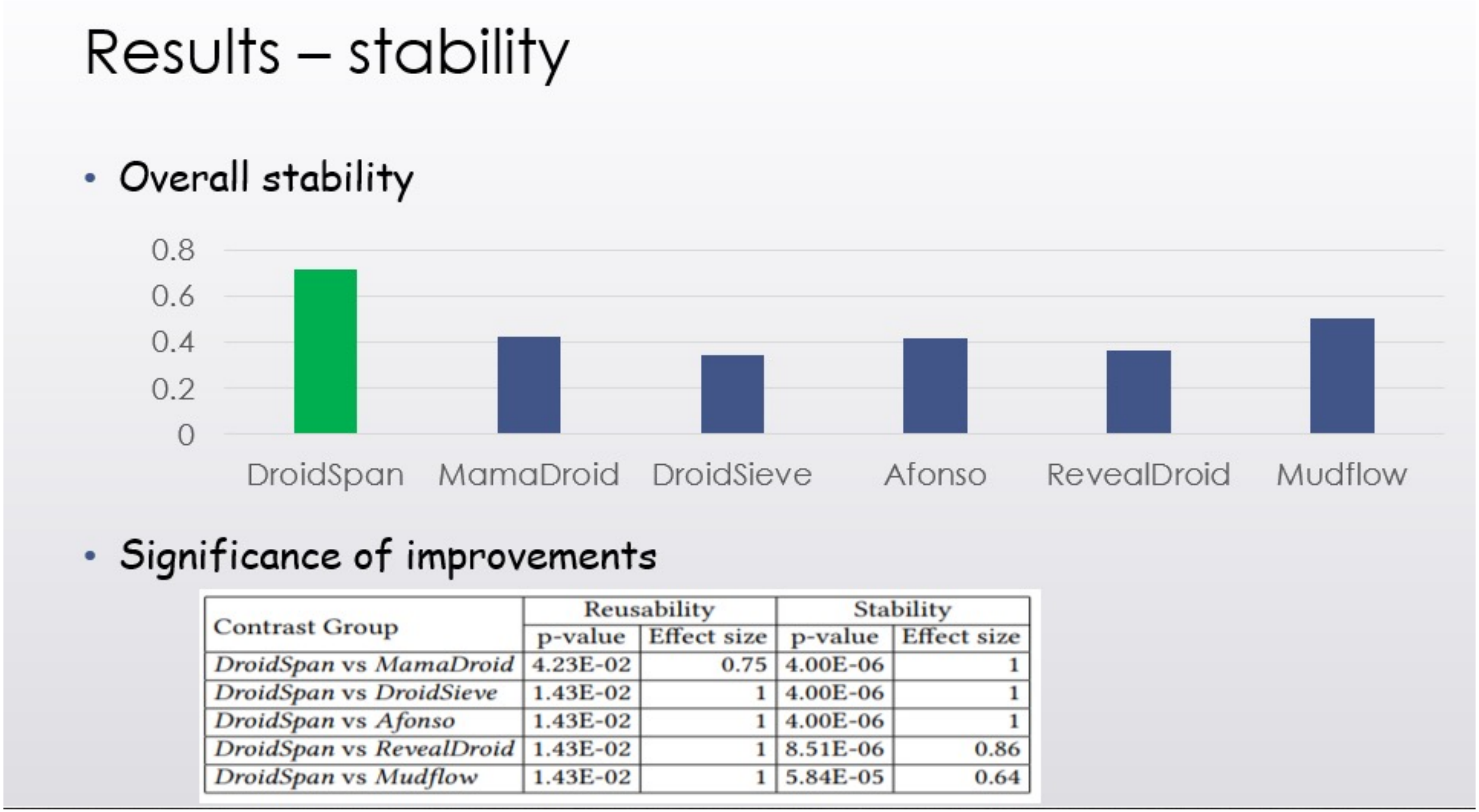
- Each dataset: 1/3 hold-out (& 10-fold CV)



Dataset	*DroidSpan*			*MamaDroid*			*DroidSieve*			*Afonso*			*RevealDroid*			*Mudflow*		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
B10+M10	0.9376	0.9360	**0.9362**	0.8424	0.8357	0.8367	0.8353	0.9347	0.8822	0.8788	0.8710	0.8718	0.8600	0.8540	0.8549	0.5246	0.5319	0.5065
B11+M11	0.9432	0.9417	0.9413	0.9893	0.9893	**0.9793**	0.9583	0.7091	0.8151	0.8978	0.8978	0.8978	0.8700	0.8641	0.8616	0.4598	0.4537	0.4563
B12+M12	0.9424	0.9424	**0.9423**	0.8378	0.8378	0.8377	0.9203	0.8000	0.8560	0.8954	0.8935	0.8935	0.8283	0.8279	0.8277	0.7344	0.6419	0.6450
B13+M13	0.9554	0.9529	**0.9525**	0.9141	0.9076	0.9060	0.9935	0.8102	0.8926	0.9217	0.9182	0.9172	0.8915	0.8823	0.8830	0.6362	0.6433	0.6311
B14+M14	0.9302	0.9272	**0.9250**	0.8462	0.8467	0.8449	0.8981	0.4528	0.6020	0.8673	0.8693	0.8665	0.8360	0.8389	0.8367	0.7040	0.7048	0.6930
B15+M15	0.9061	0.9042	**0.9036**	0.8450	0.8440	0.8442	0.8162	0.9193	0.8647	0.7798	0.7610	0.7514	0.8236	0.8014	0.7939	0.7213	0.7218	0.7125
B16+M16	0.9352	0.9342	**0.9339**	0.9021	0.8969	0.8955	0.8275	0.9787	0.8968	0.8138	0.8068	0.8025	0.8660	0.8444	0.8389	0.7532	0.5936	0.6135
B17+M17	0.9723	0.9720	**0.9720**	0.9126	0.9093	0.9098	0.8910	0.8892	0.8891	0.9510	0.9493	0.9493	0.8546	0.8360	0.8334	0.8331	0.7105	0.6668
Average	0.9408	0.9393	**0.9388**	0.8835	0.8810	0.8794	0.8929	0.7956	0.8271	0.8780	0.8738	0.8719	0.8523	0.8431	0.8408	0.6761	0.6284	0.6185

DroidSpan achieved reusability of 94% with small variations across years, outperforming all the five baselines considered (by 6–32%).

DroidSpan - a detector based on SAD profiles



Results – stability

- Overall stability

- Significance of improvements

Contrast Group	Reusability		Stability	
	p-value	Effect size	p-value	Effect size
DroidSpan vs MamaDroid	4.23E-02	0.75	4.00E-06	1
DroidSpan vs DroidSieve	1.43E-02	1	4.00E-06	1
DroidSpan vs Afonso	1.43E-02	1	4.00E-06	1
DroidSpan vs RevealDroid	1.43E-02	1	8.51E-06	0.86
DroidSpan vs Mudflow	1.43E-02	1	5.84E-05	0.64

References

- Amir Houmansadr, CS660: Advanced Information Assurance, Spring 2015
- John Mitchell, CS155, Spring 2017
- Dominic Chen, Introduction to Mobile Security
- Jiaojiao Fu, Detecting and Preventing Privilege Escalation, 2013
- Xinming Ou, Android System Security
- Collin Donaldson, Android
- Vitaly Shmatikov, CS6431, Security of Mobile Applications
- Yinshu Wu, Understanding Android security
- Cai, Haipeng, Na Meng, Barbara Ryder, and Daphne Yao. "Droidcat: Effective android malware detection and categorization via app-level profiling." IEEE Transactions on Information Forensics and Security 14, no. 6 (2018): 1455-1470.
- Cai, Haipeng. "Assessing and improving malware detection sustainability through app evolution studies." ACM Transactions on Software Engineering and Methodology (TOSEM) 29, no. 2 (2020): 1-28.

Summary

- Android: dominating mobile app and malware market
- Android security mechanisms: sandoxing and permission-base access control
  - Inter-app communication offers flexibility/reuse, also increasing attack surface
- Vulnerabilities facilitates/enables attacks, leading to broad security consequences
- Defense strategies: analyzing code behaviors, learning malicious patterns
- Sustainable solutions: tackling app/malware evolution (moving target)