Discovering Flaws in Security-Focused Static Analysis Tools for Android using Systematic Mutation

Richie Bonett,
Kaushal Kafle,
Kevin Moran,
Adwait Nadkarni &
Denys Poshyvanyk

Friday, Aug 17th, 2018

WILLIAM & MARY
CHARTERED 1693

Motivation

Motivation


Taintdroid
MalloDroid
Flowdroid
DroidSafe
BlueSeal
DidFail
IccTA
Argus-SAF
CryptoLint
Safer Apps
End User

SECURITY ANALYSIS OF APPLICATIONS

- Security tools have diverse security goals
- Security analysis of apps is highly beneficial to end users
  - Keeps the ecosystem clear of malicious or vulnerable apps

Detecting SSL Vulnerabilities

OAuth-token Tracking

Data Leak Detection

Permission misuse

Intent spoofing

Password tracking

SECURITY ANALYSIS OF APPLICATIONS

- Security tools have diverse security goals

Detecting SSL Vulnerabilities

OAuth-token Tracking

Data Leak Detection

Q: Do we really know how well these tools work?

beneficial to end users

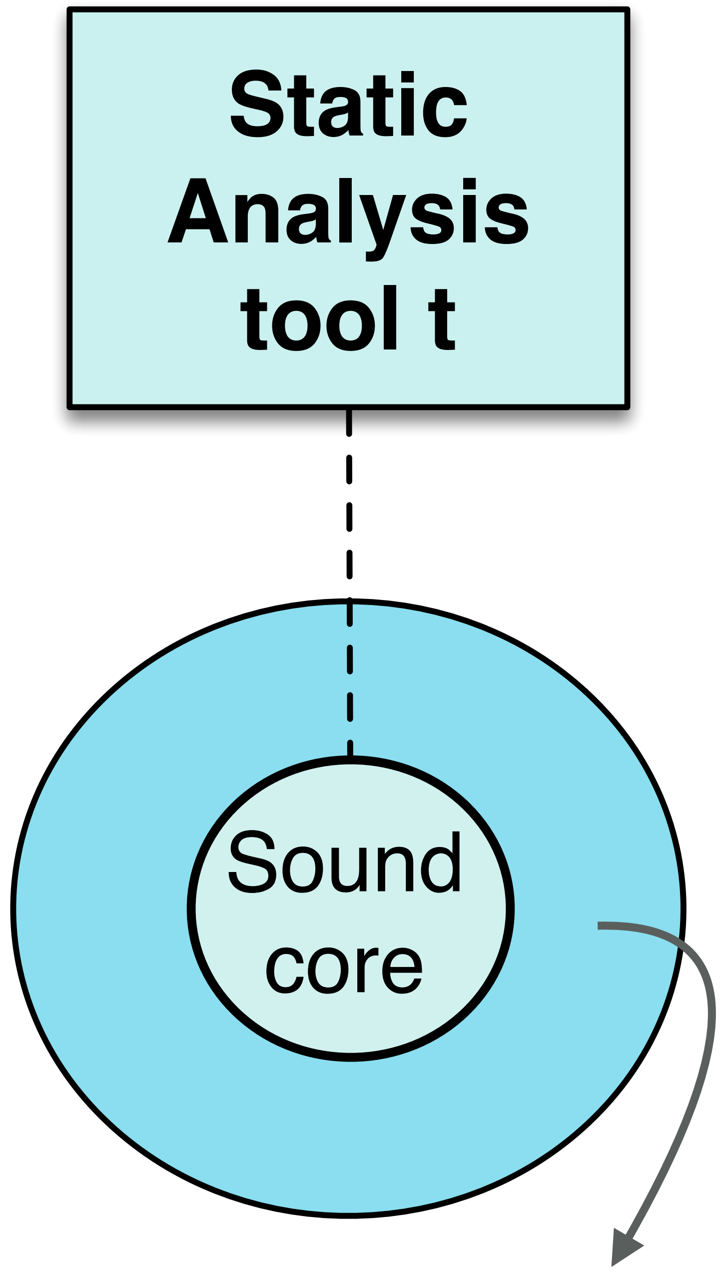
spoofing

Password tracking

- Keeps the ecosystem clear of malicious or vulnerable apps

SOUNDINESS

- 2015: *Soundiness* manifesto[1]
- Static analysis tools are implicitly expected to be sound (i.e., they over-approximate)
- In practice, all tools are ***soundy***: A sound core, but with some unsound assumptions to be practical; e.g. JNI, Reflection
- Soundy tools are practical
- *However,* developers might not document unsound choices for various reasons



Java Reflection

Dynamic code loading

JNI …

[1] Livshits, Benjamin, et al. "In defense of soundiness: a manifesto." *Communications of the ACM* 58.2 (2015): 44-46.

Soundiness

- 2015: *Soundiness* manifesto[1]

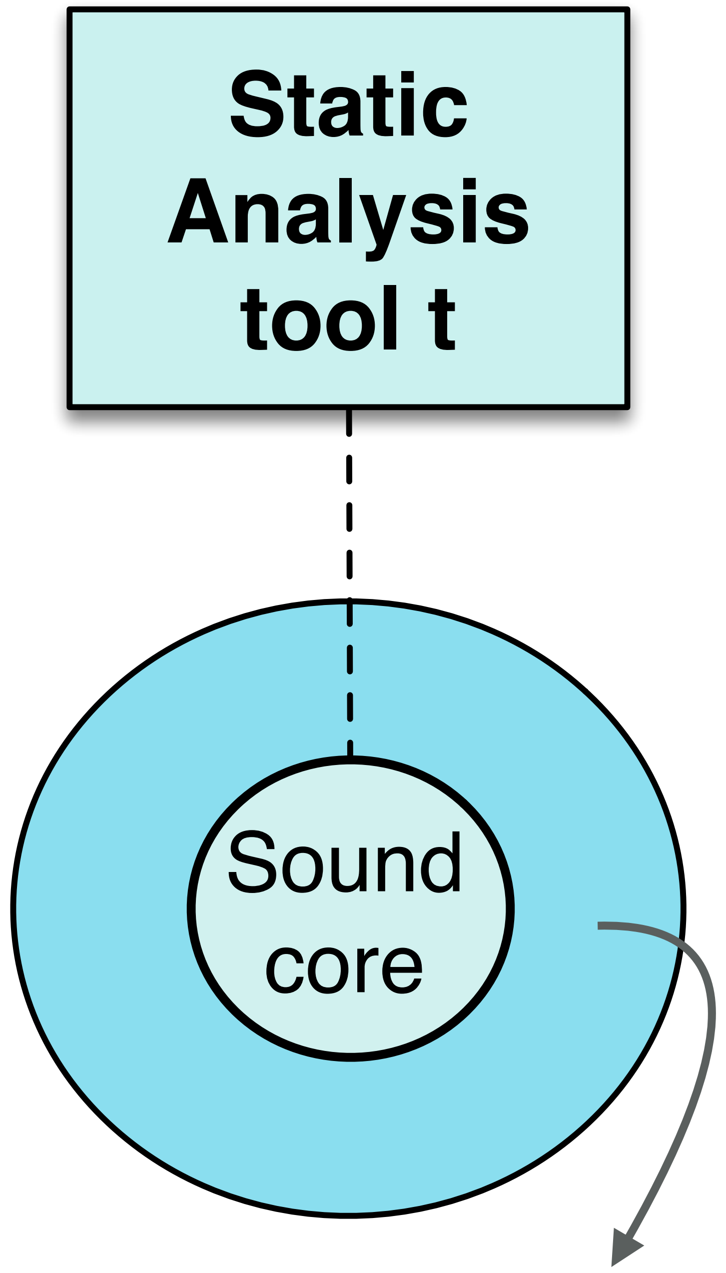
- Static analysis tools are implicitly expected to be sound (i.e., they over-approximate)

- In practice, all tools are ***soundy***: A sound core, but with some unsound assumptions to be practical; e.g. JNI, Reflection

- Soundy tools are practical

- *However,* developers might not document unsound choices for various reasons

- ***We want to discover the extent of the unsound decisions***



Java Reflection

Dynamic code loading

JNI ...

[1] Livshits, Benjamin, et al. "In defense of soundiness: a manifesto." *Communications of the ACM* 58.2 (2015): 44-46.

SOUNDINESS OF MOBILE SECURITY TOOLS

- The scope of the soundiness manifesto is language features
- We target security analysis of mobile apps (e.g., data leak detection, SSL vuln, etc.)
- This paper: A general discussion on the *design/ implementation choices* in the context of the target platform, i.e., Android, and its unique abstractions:
  - Application model
  - Inter-component communication
  - Asynchronous invocation and component lifecycles

Intent messages

BroadcastReceiver

Callbacks

Activities

Fragments

XML Resource Files

OUR VISION

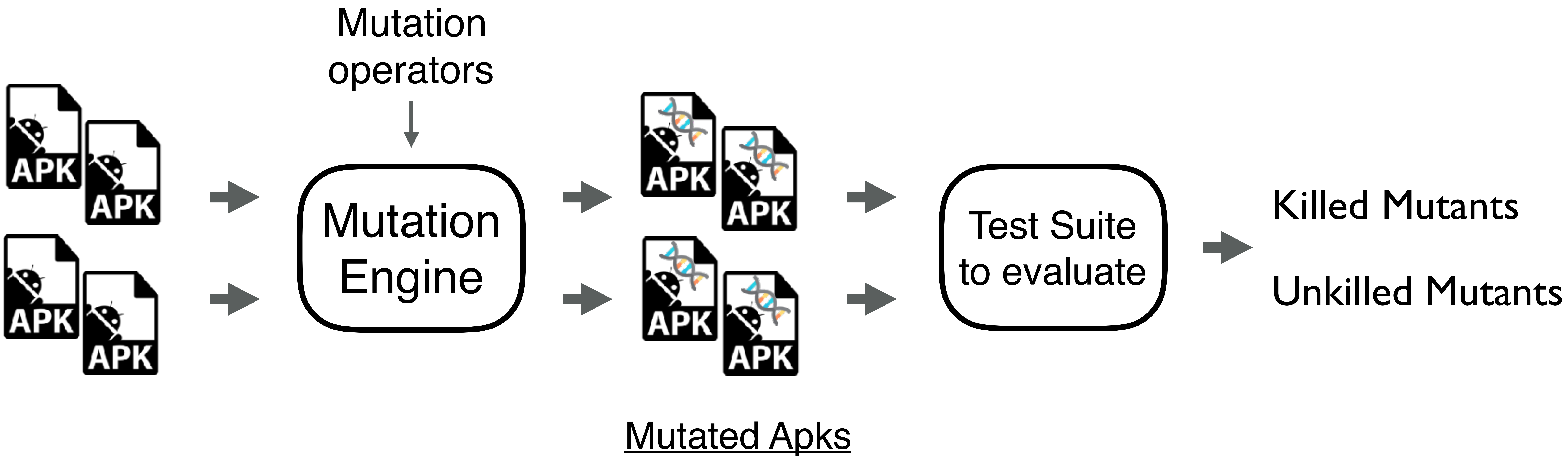
A framework that enables systematic evaluation of existing security tools to identify and document unsound decisions, eventually expanding the sound core

- Benefits:
  - **Researchers**: *discover* undocumented flaws in tools
  - **Developers**: *build* more effective tools by discovering easily fixed but evasive bugs
  - **Users**: *benefit* from better detection, and hence a better application ecosystem

μSE: Mutation-based Soundness Evaluation

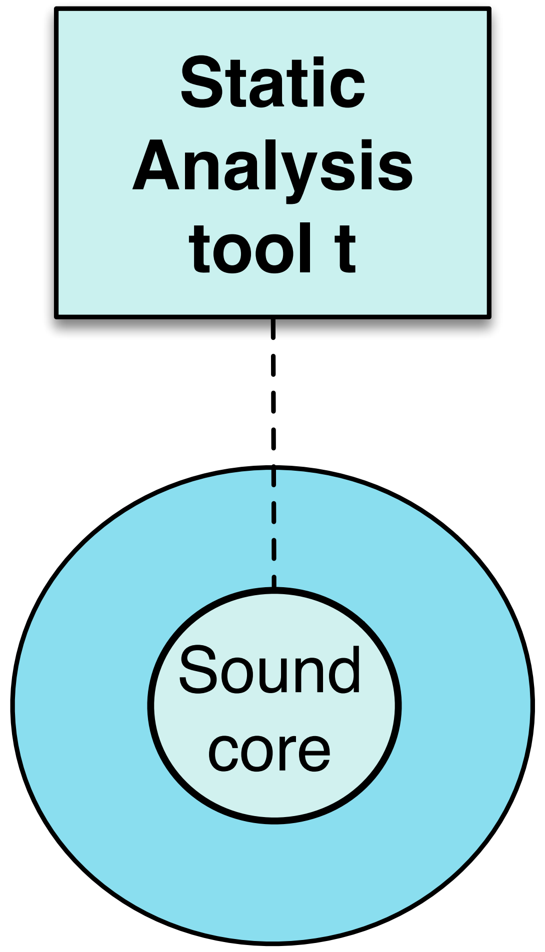
- μSE leverages *mutation analysis* for systematic evaluation of security tools
- **Contextualizes** mutation analysis to security
- Develops the abstractions of
    1. Security operators and
    2. Mutation schemes

MUTATION ANALYSIS BACKGROUND



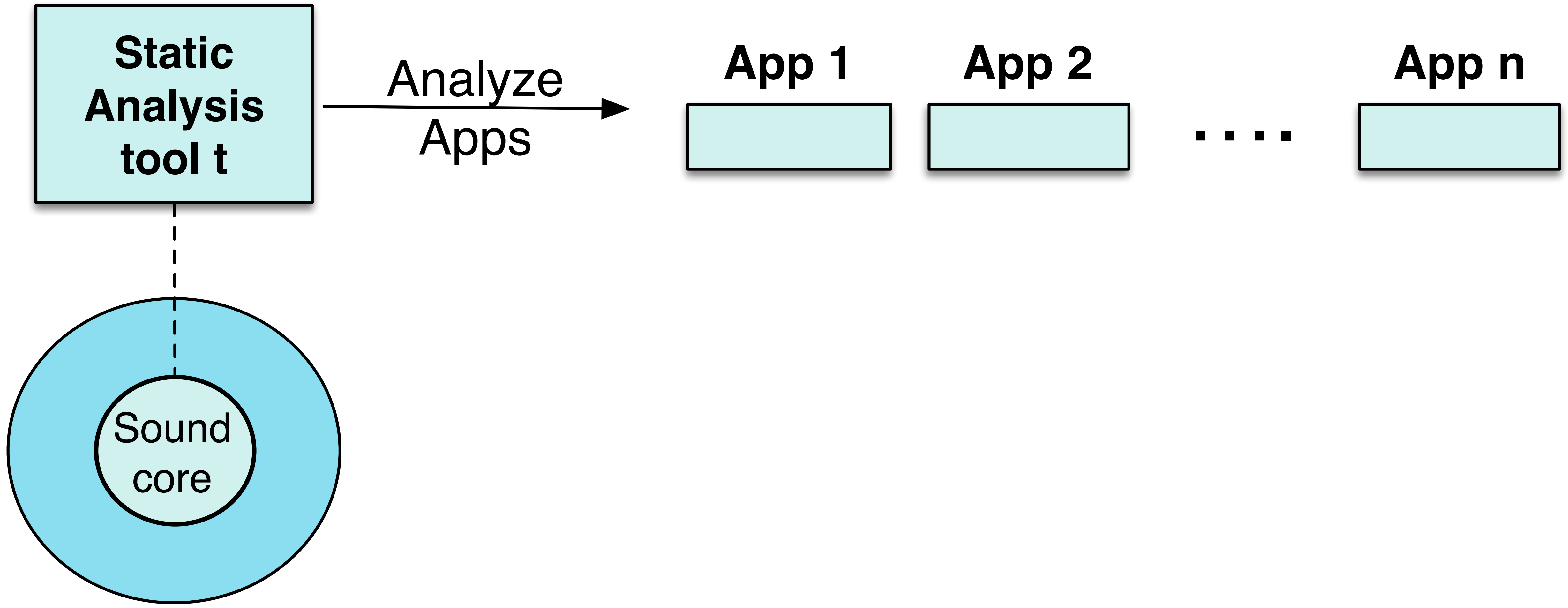
μSE Overview

- μSE leverages *mutation analysis* for systematic evaluation of security tools



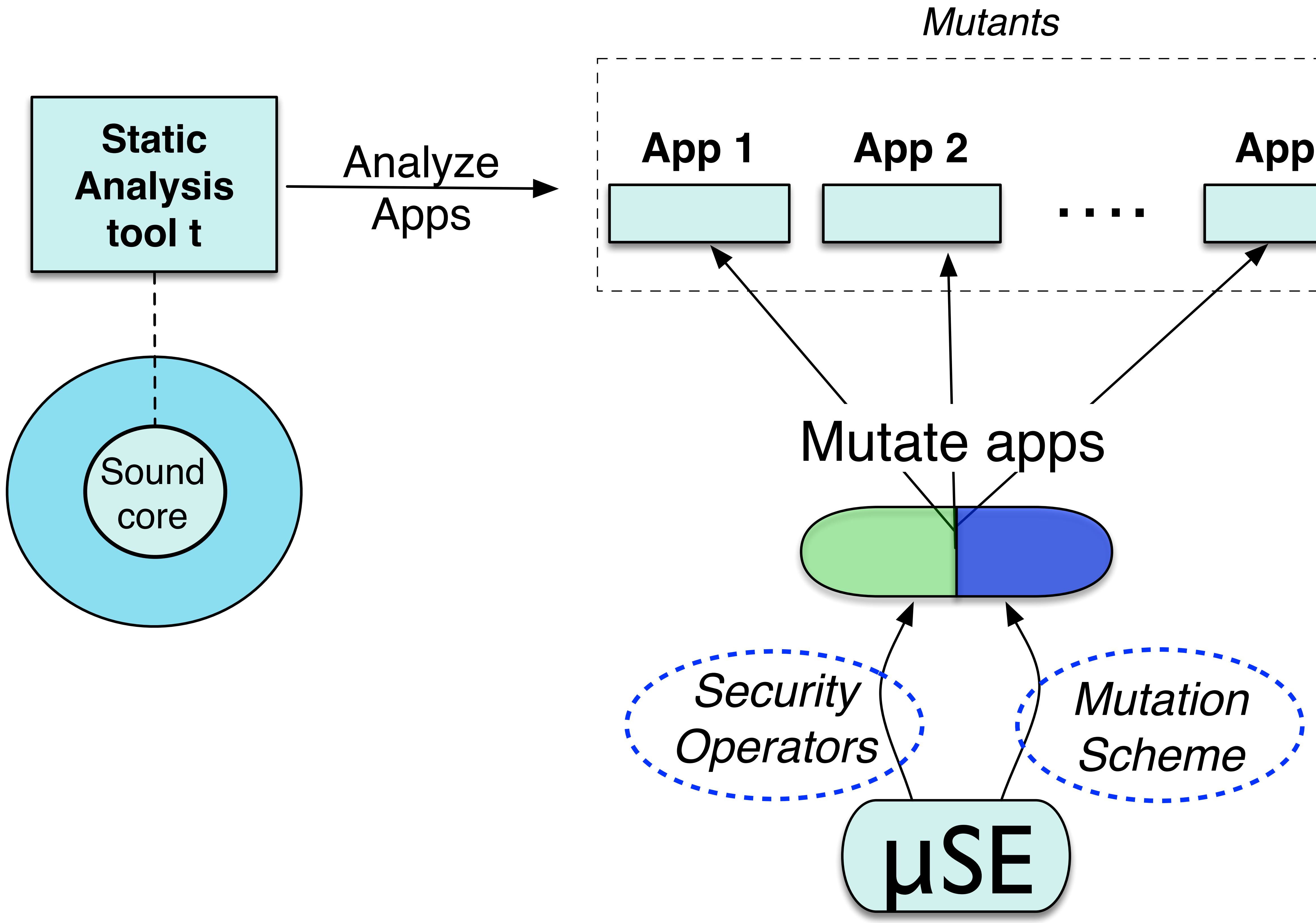
µSE OVERVIEW

- µSE leverages *mutation analysis* for systematic evaluation of security tools



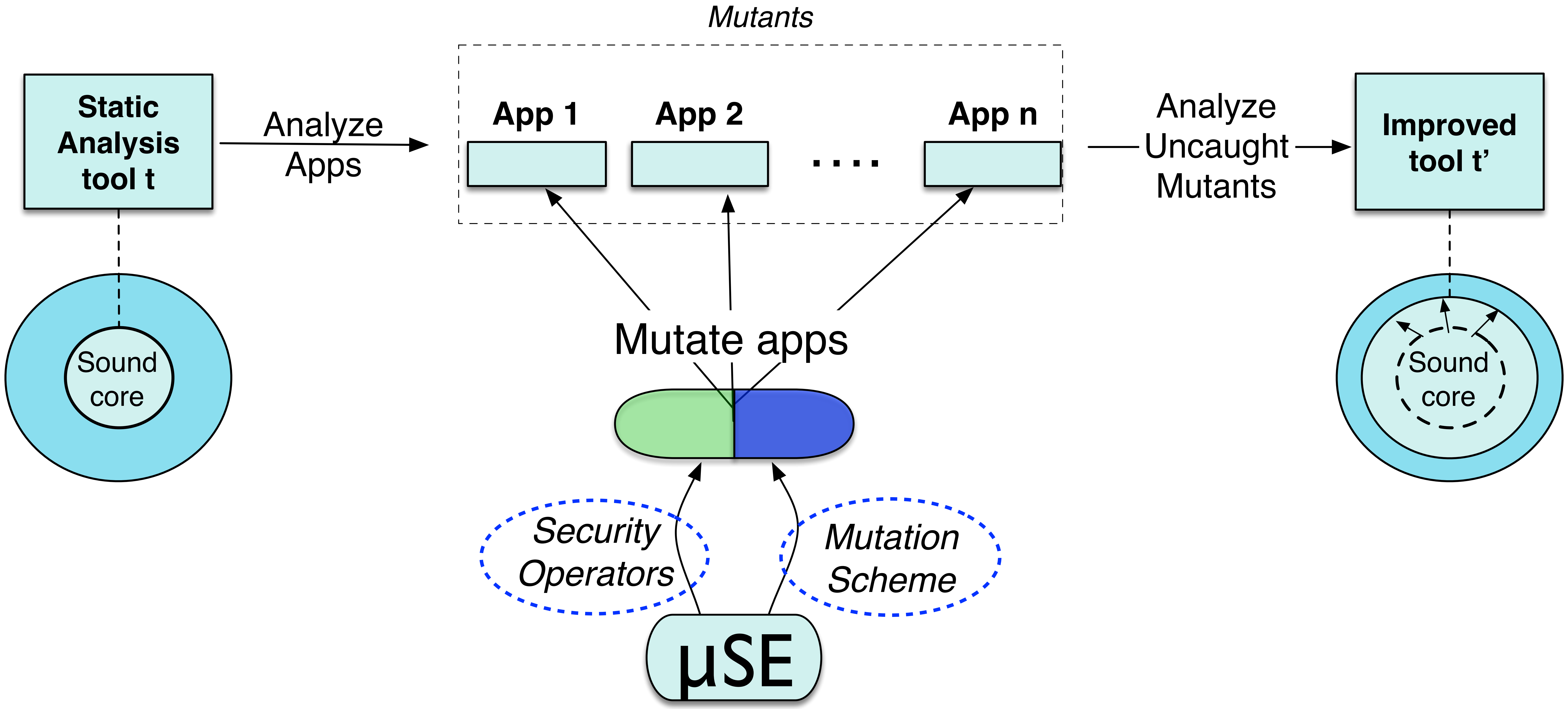
μSE Overview

- μSE leverages *mutation analysis* for systematic evaluation of security tools



µSE Overview

- µSE leverages *mutation analysis* for systematic evaluation of security tools



μSE DESIGN

- Basic Components and their definitions:
  - Security operator: What anomaly/mutation to insert in the app
  - Mutation scheme: Where to place/seed it

µSE DESIGN: SECURITY OPERATORS

- Challenges:
  - Too fine-grained —> Not scalable
  - Too generic —> Not effective as different tools have different security focus
- µSE defines security operator in terms of *security goals* of the tools
  - Scalable to tools with similar security goals (e.g., data leak detection)

μSE DESIGN: SECURITY OPERATORS

1. Operator for data leak detectors

```
dataLeak = Location.read()

log.d(dataLeak)
```

2. Operator for SSL vulnerability detectors

```
boolean isServerTrusted() {

return true }
```

μSE DESIGN: MUTATION SCHEME

- Multiple strategies with different objectives
  1. Reachability analysis
  2. Android Abstractions
  3. Security goals

μSE Design: Mutation scheme

1. Reachability analysis

- Placing operator at the start of every method
  - Helps in the evaluation of the coverage of flaws
- Simplest mutation scheme for operator placement

μSE DESIGN: MUTATION SCHEME

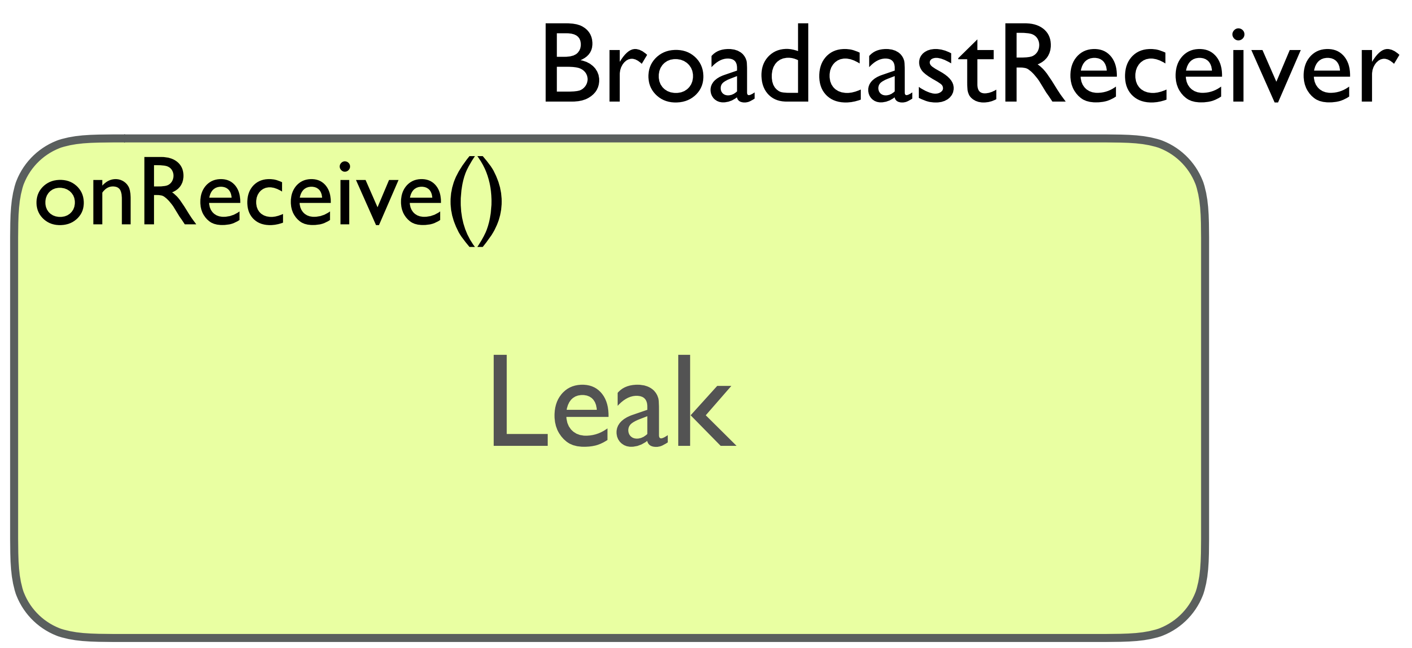
2. Android abstractions

- Model unique aspects of Android platform
- Mutants are built *specifically for Android* by choosing its unique abstractions as the starting point
  - Activity & Fragment Lifecycles
  - Callbacks
  - Intent messages
  - Android Resource files

µSE Design: Mutation scheme

2. Android abstractions

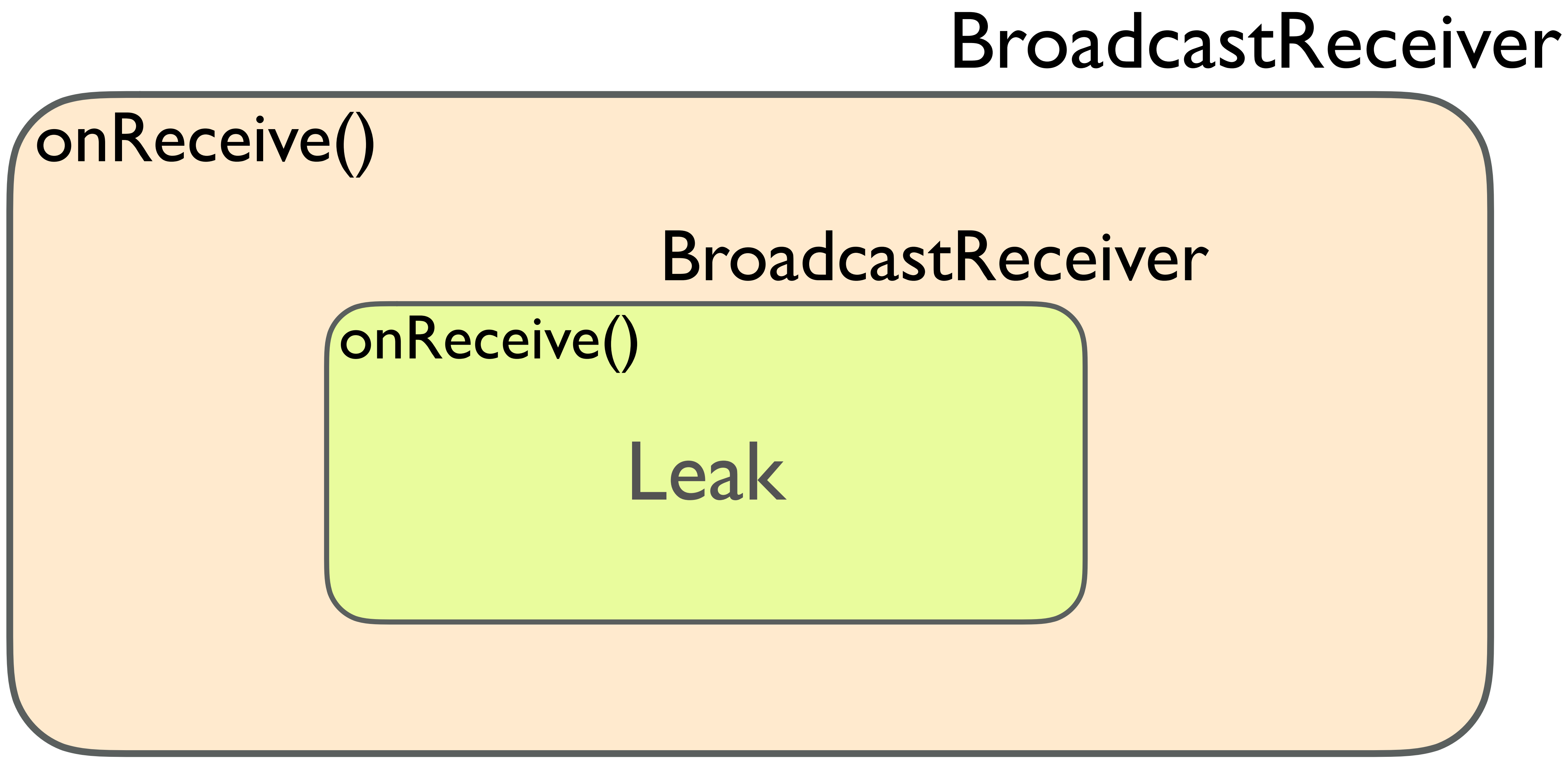
- Model unique aspects of Android platform
- Mutants are built *specifically for Android* by choosing its unique abstractions as the starting point
  - Activity & Fragment Lifecycles
  - Callbacks
  - Intent messages
  - Android Resource files



µSE Design: Mutation scheme

2. Android abstractions

- Model unique aspects of Android platform
- Mutants are built *specifically for Android* by choosing its unique abstractions as the starting point
  - Activity & Fragment Lifecycles
  - Callbacks
  - Intent messages
  - Android Resource files



3. Security goal

- Accounting for the specific objective of the tool under scrutiny
- Can be applied to other tools with similar goals
- *E*.g., a **taint-based** scheme for data leak detection tools

μSE Design: Mutation scheme

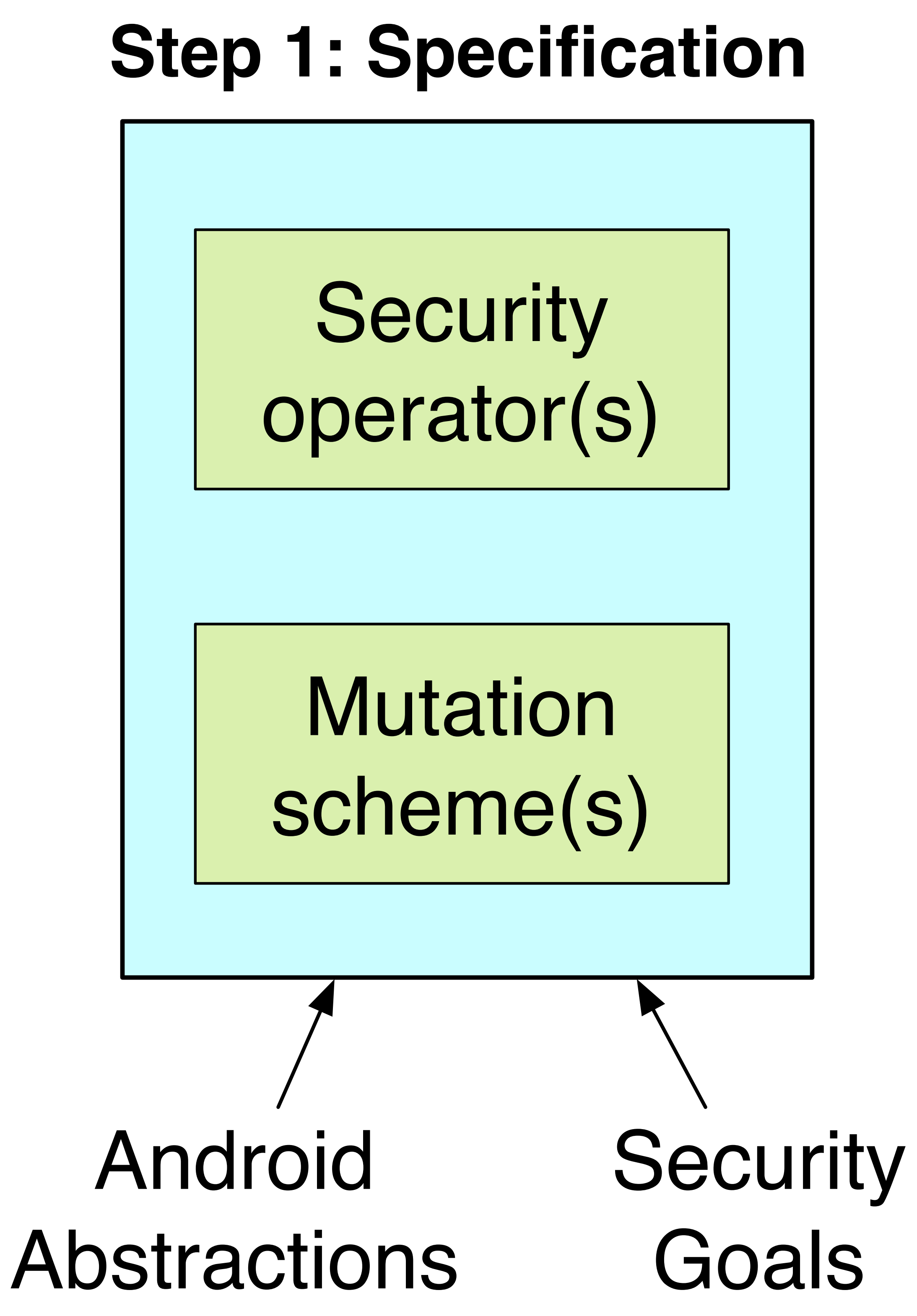
3. Security goal

- Accounting for the specific objective of the tool under scrutiny
- Can be applied to other tools with similar goals
- *E*.g., a **taint-based** scheme for data leak detection tools

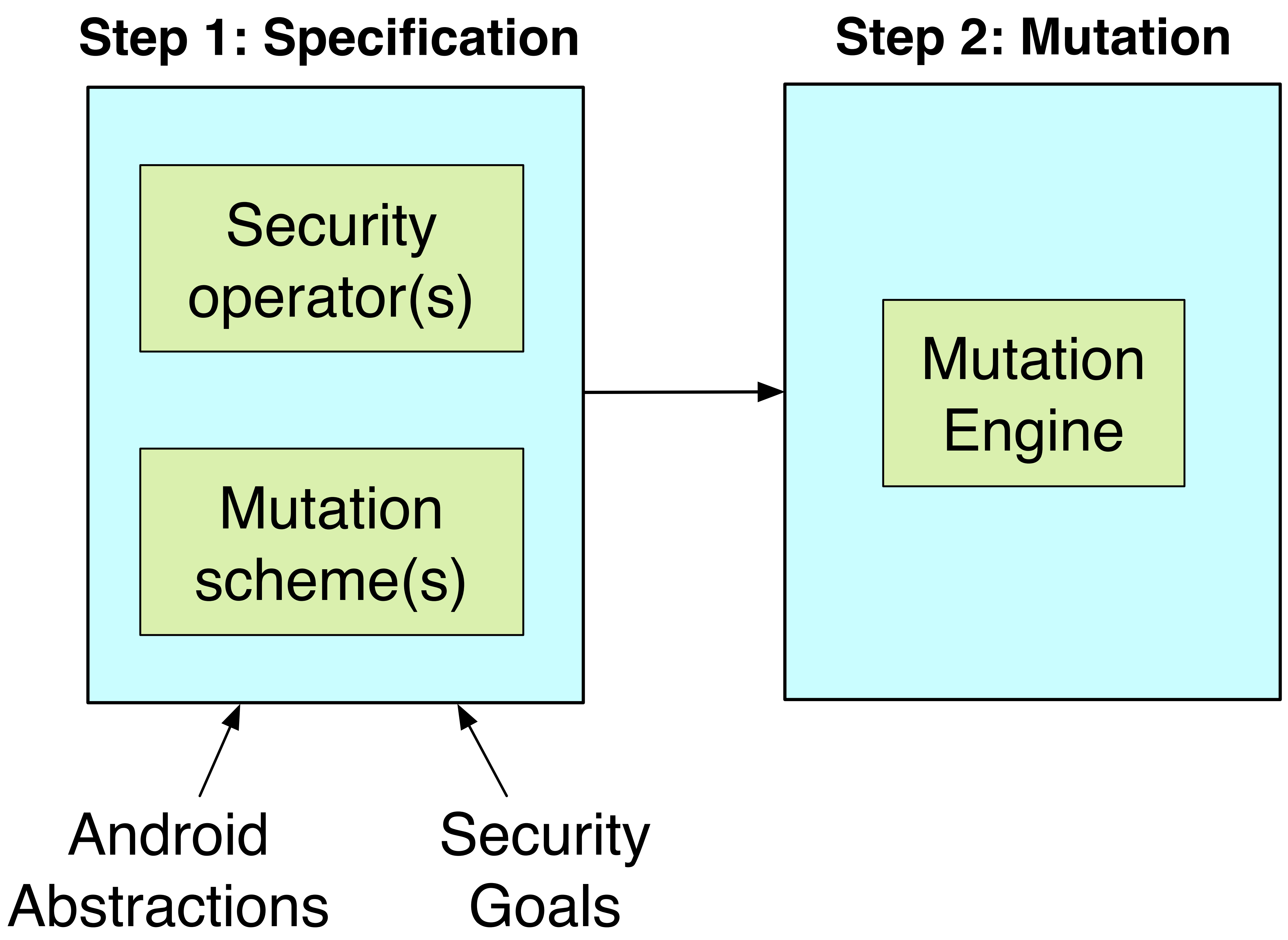
Source $\downarrow$ onStart()

Sink $\downarrow$ onResume()

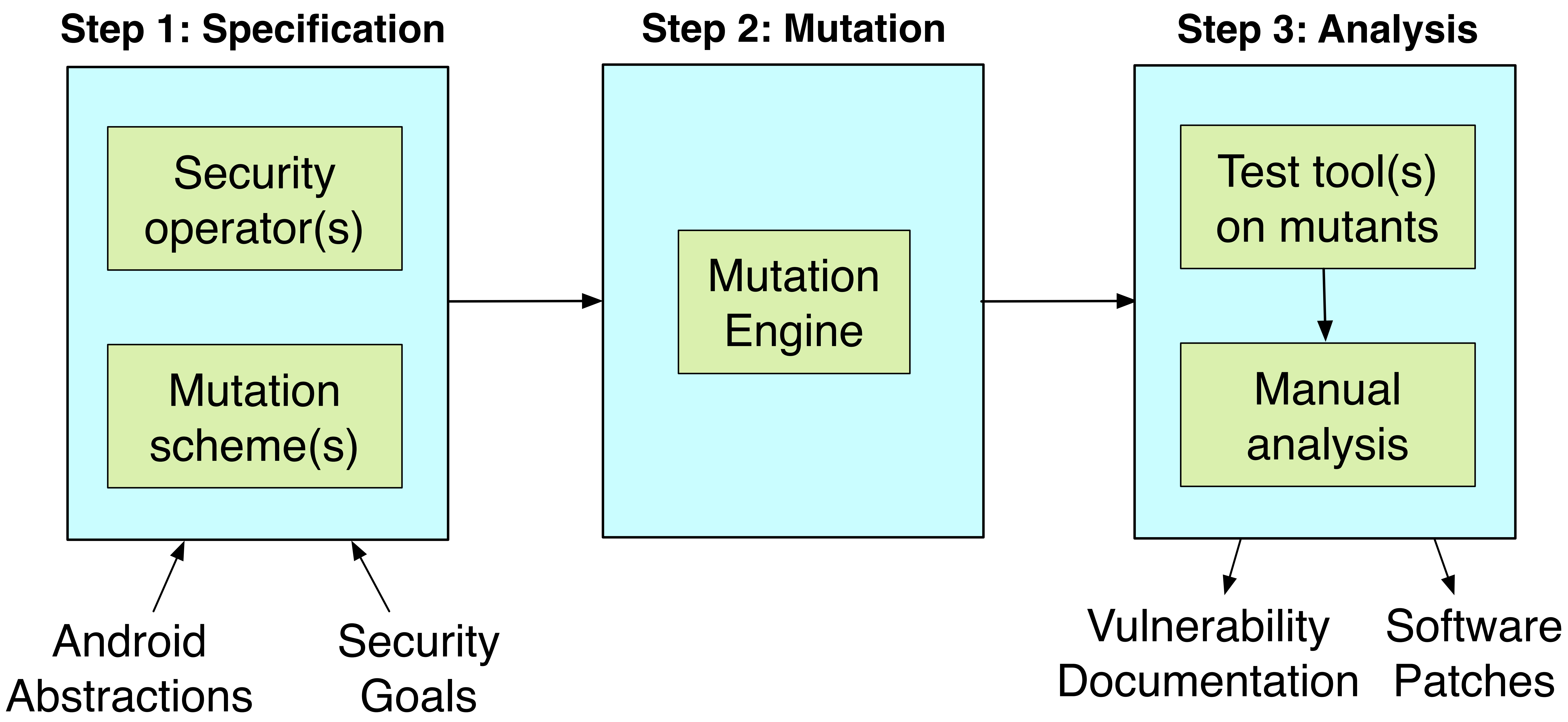
IMPLEMENTATION



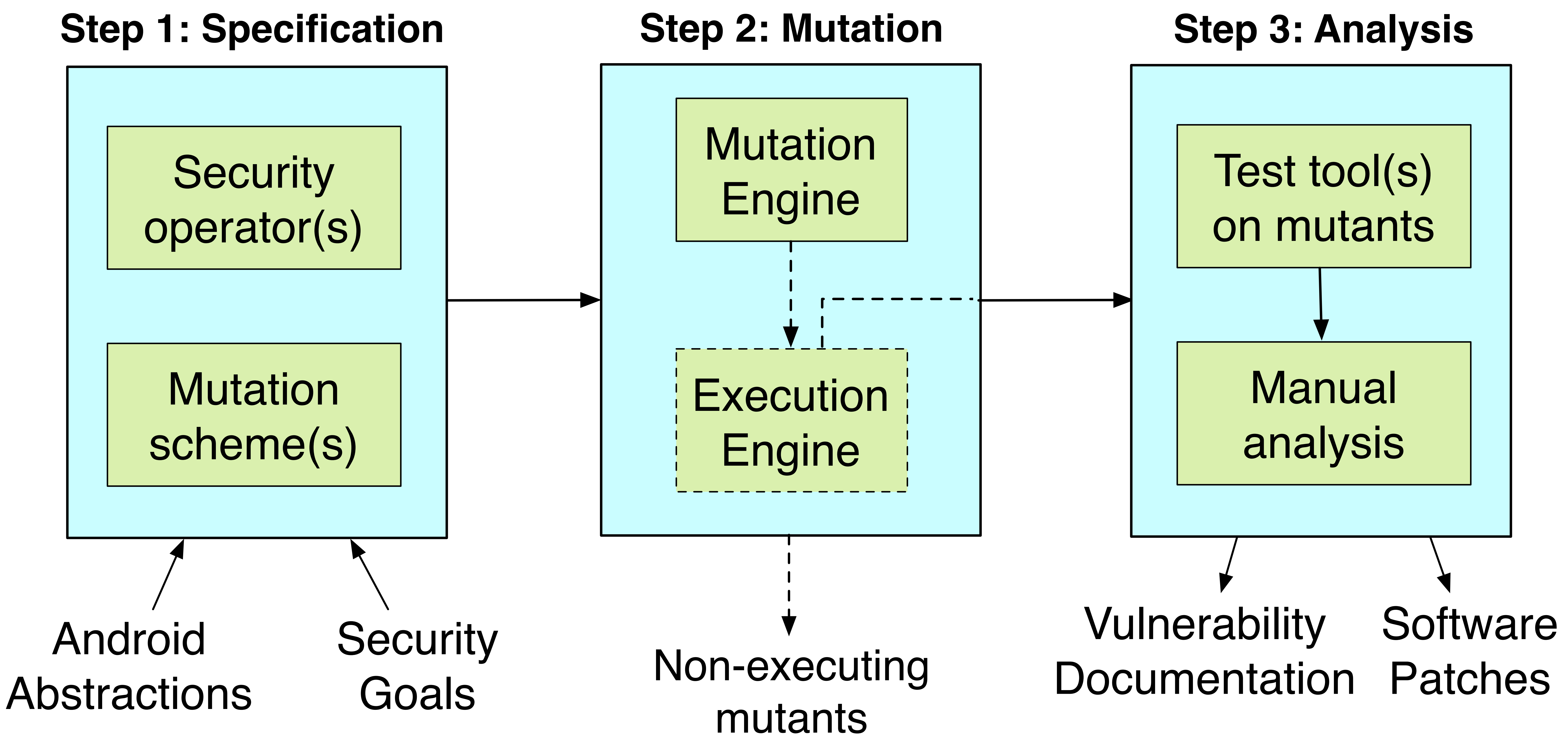
IMPLEMENTATION



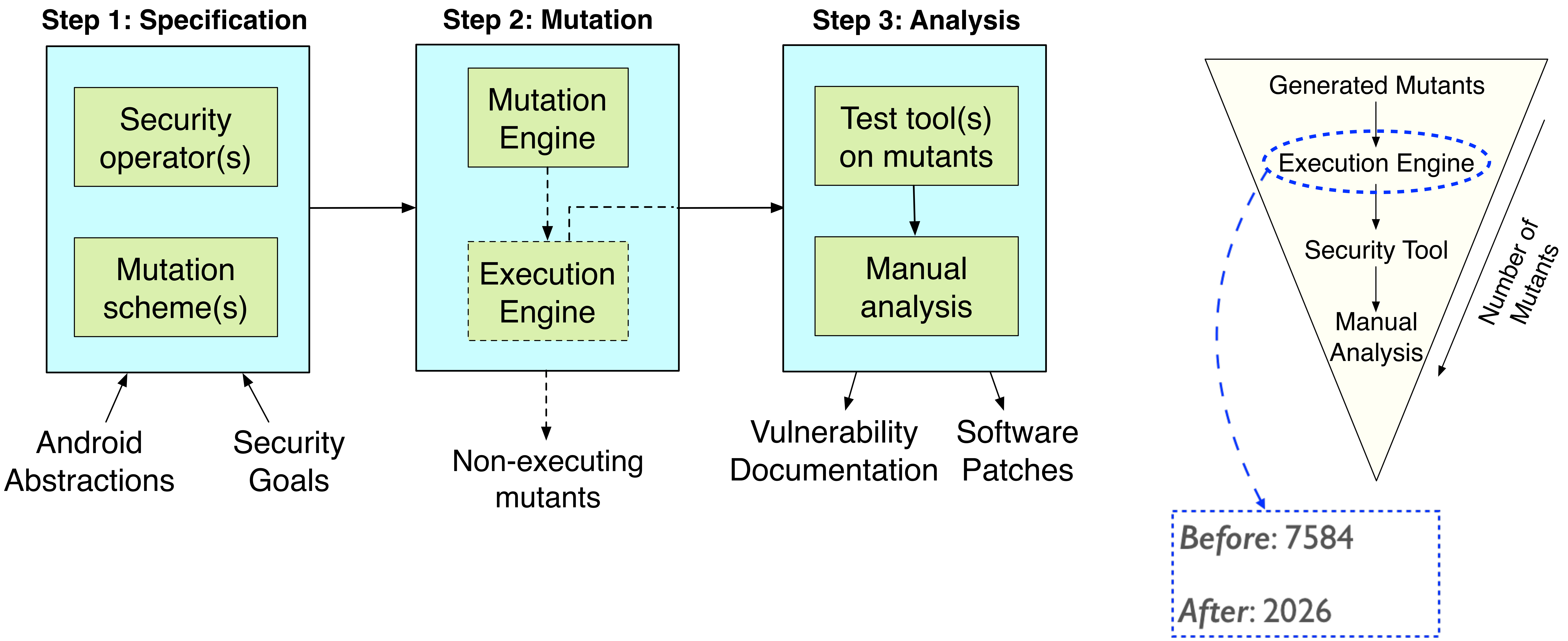
IMPLEMENTATION



IMPLEMENTATION



IMPLEMENTATION



EVALUATION

- We evaluate the effectiveness of µSE using a case study
- Security goal we chose for our case study: *Data leak detection*

EVALUATION: TESTING DATA LEAK DETECTORS

- 7,584 mutants in total, 2,026 verified as executable
- 3 data leak detection tools evaluated using 2,026 mutants

TOOLS	UNDETECTED LEAKS
Flowdroid 2.0	987/2026 (48.7%)
Argus-SAF	1480/2026 (73.1%)
DroidSafe	83/2026 (4.1%)

EVALUATION: TESTING DATA LEAK DETECTORS

- 7,584 mutants in total, 2,026 verified as executable
- 3 data leak detection tools evaluated using 2,026 mutants

TOOLS	UNDETECTED LEAKS
Flowdroid 2.0	987/2026 (48.7%)
Argus-SAF	1480/2026 (73.1%)
DroidSafe	83/2026 (4.1%)

- **Impact**: Cited over 900 times
- **Immediate response and benefits**: Flowdroid is actively being maintained

Evaluation: Flaws Discovered

	Vulnerability Class (VC)	Example flaw in VC	Description of the flaw
1	**Missing Callbacks (*5 flaws*)**	Fragments	Doesn't model Android Fragments correctly.
2	**Missing Implicit Calls (2 *flaws*)**	RunOnUIThread	Misses a path to Runnable.run()for runnables passed into Activity.runOnUIThread()
3	**Incorrect Modeling of Anonymous Classes (2 *flaws*)**	BroadcastReceiver	Misses the onReceive() callback of a BroadcastReceiver implemented programmatically and registered within another programmatically defined BroadcastReceiver's onReceive() callback.
4	**Incorrect Modeling of Asynchronous Methods (4 *flaws*)**	LocationListenerTaint	Misses the flow from a source in the onStatusChanged() callback to a sink in the onLocationChanged() callback of the LocationListener interface, despite recognizing leaks wholly contained in either.

EVALUATION: FLAW PROPAGATION

	FLAW	FD 2.5.1	FD 2.5.0	FD 2.0	BLUESEAL	ICCTA	HORNDROID	ARGUS	DROIDSAFE	DIDFAIL
1	DialogFragmentShow	✘	✘	✘	✓	✘	✘	✓	✓	✘
2	PhoneStateListener	✘	✘	✘	✓	✘	✘	✓	✓	✘
3	NavigationView	✘	✘	✘	-	✘	-	✘	-	✘
4	SQLiteOpenHelper	✘	✘	✘	✓	✘	✘	✘	✓	✘
5	Fragments	✘	✘	✘	✘	✘	✘	✘	-	✘
6	RunOnUIThread	✘	✘	✘	✓	✘	✘	✘	✓	✘
7	ExecutorService	✘	✘	✘	✓	✘	✘	✘	✓	✘
8	ButtonOnClickToDialogOnClick	✘	✘	✘	✓	✘	✓	✓	✘	✘
9	BroadcastReceiver	✘	✘	✘	✓	✘	✓	✓	✓	✘
10	LocationListenerTaint	✘	✘	✘	✓	✘	✓	✓	✓	✘
11	NSDManager	✘	✘	✘	✓	✘	✓	✘	✓	✘
12	ListViewCallbackSequential	✘	✘	✘	✓	✘	✓	✓	✓	✘
13	ThreadTaint	✘	✘	✘	✓	✘	✓	✓	✓	✘

✘ - Fails to detect

- ***Inheriting flowdroid as a black box*** - IccTA (13/13), DidFail (13/13)
- ***Motivated by flowdroid's design*** (but augmented to their need) - Argus-SAF (6/13)
- ***Implementing their own methodologies*** - BlueSeal (1/13), HornDroid (6/13), DroidSafe (1/13)

Recall: Expanding the sound core

- We could fix *one* of the problems (fragment, FlowDroid 2.0)
- However, fixing flaws is significantly challenging
  - Some flaws are design-choices that are hard to immediately fix (e.g. Runnable)
  - Some are unsolved research challenges (e.g., BroadcastReceiver)
- *μSE effectively serves the function of discovering/documenting these for future research*

CAVEATS

- μ*SE* doesn't claim soundness
  - Aims to increase the confidence in the results of *soundy* tools by discovering and documenting unsound choices
- μSE doesn't replace formal verification
  - Rather a framework for systematically uncovering flaws in security tools
  - Significant advancement over manually curated toolkits

CONCLUDING REMARKS

- µSE demonstrates the effectiveness of mutation analysis at discovering undocumented flaws in security tools
- Flaws not only affect individual tools, but propagate to future research
- Android evolves, and µSE is a significant improvement over manually curated benchmarks that need keep up with Android's fast-paced evolution
- µSE allows patching of easily fixable but evasive flaws; however, this is a hard problem in general

Code and data at:

https://muse-security-evaluation.github.io/

Thank you!

Kaushal Kafle
William & Mary
kkafle@cs.wm.edu