UIC UNIVERSITY OF ILLINOIS AT CHICAGO

AppSeer: Discovering Interface Defects among Android Components

Vincenzo Chiaramida, Francesco Pinci, Ugo Buy and Rigel Gjomemo
University of Illinois at Chicago
4 September 2018

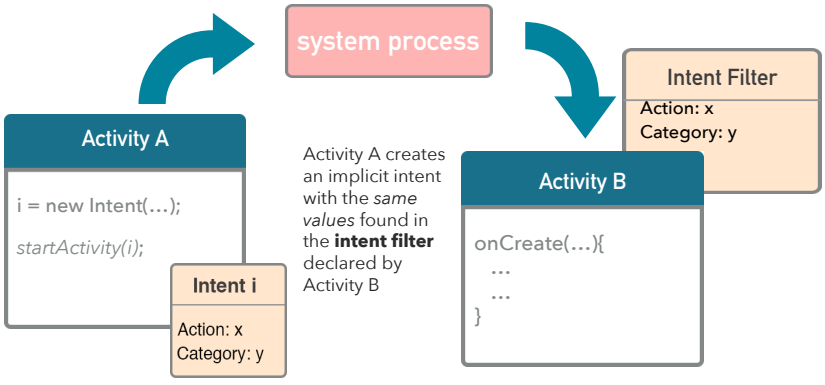
Slides by: Vincenzo Chiaramida
Presented by: Ugo Buy

1. GOALS and MOTIVATION

RESEARCH GOALS

- Analyze interactions among components in Android apps
- Discover interface defects
- Develop techniques and tools for automatic defect detection

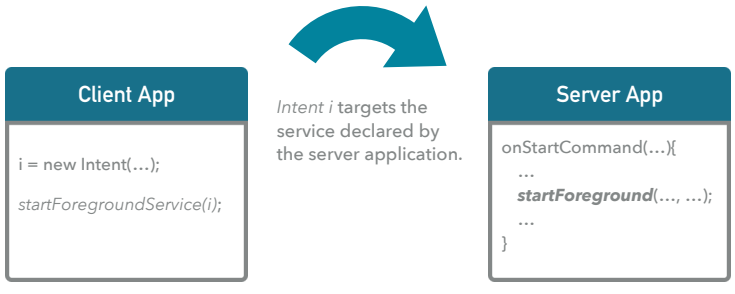
MOTIVATION



2. FOREGROUND SERVICES

NEW CLIENT-SIDE OPTION TO START SERVICE

- Oreo and Pie have new client-side method: *startForegroundService(Intent)*
- … but then service must call *startForeground(int, Notification)*



CLIENT AND SERVICE-SIDE COMBINATIONS

CLIENT APP	SERVER APP	RESULT
startService()	onStartCommand()	Background Service

2. FOREGROUND SERVICES
ANALYSIS DETECTION

CLIENT AND SERVICE-SIDE COMBINATIONS

CLIENT APP	SERVER APP	RESULT
startService()	onStartCommand()	Background Service
startService()	onStartCommand() + startForeground()	Foreground Service

2. FOREGROUND SERVICES

ANALYSIS DETECTION

CLIENT AND SERVICE-SIDE COMBINATIONS

CLIENT APP	SERVER APP	RESULT
startService()	onStartCommand()	Background Service
startService()	onStartCommand() + startForeground()	Foreground Service
startForegroundService()	onStartCommand() + startForeground()	Foreground Service

2. FOREGROUND SERVICES

ANALYSIS DETECTION

CLIENT AND SERVICE-SIDE COMBINATIONS

CLIENT APP	SERVER APP	RESULT
startService()	onStartCommand()	Background Service
startService()	onStartCommand() + startForeground()	Foreground Service
startForegroundService()	onStartCommand() + startForeground()	Foreground Service
startForegroundService()	onStartCommand()	

NEW DESIGN CHOICES

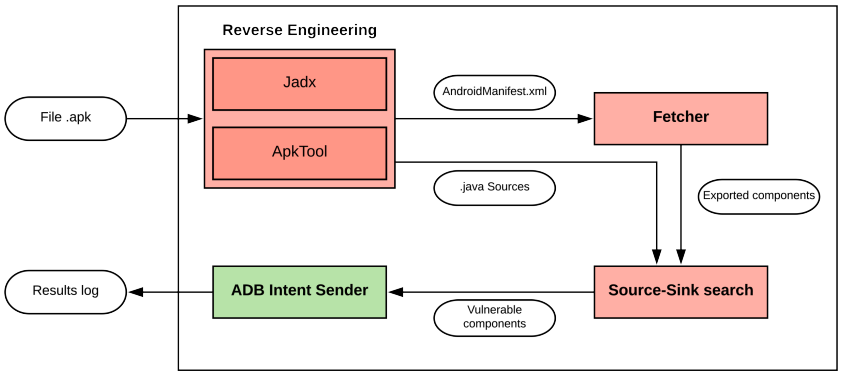
- Client side: Use *startService()* or *startForegroundService()*?
- Server side: *startForeground()* – To call or not to call?

2. FOREGROUND SERVICES
ANALYSIS DETECTION

APPSEER: DETECTION TOOL

- Goal #1: Classify services within applications as safe or vulnerable
- Goal #2: Evaluate the obtained results

APPSEER ARCHITECTURE



2. FOREGROUND SERVICES
ANALYSIS DETECTION

PHASE 3: SOURCE-SINK SEARCH

- Input: list of exported services, obfuscated source files
- Outputs: list of *vulnerable* services
- Objective: examine the *hierarchy* of the component under investigation, trying to find a flow from source to sink, without code execution

2. FOREGROUND SERVICES

ANALYSIS | DETECTION

- Hierarchy H of a service Σ:

H is a set of classes: $H = \{ \Gamma_1, \Gamma_2 \ldots, \Gamma_n \}$

$\Gamma_1 \in$ {"Service", "IntentService"}

$\Gamma_n = \Sigma$

$\forall\, i \in \{1, \ldots, n\text{-}1\}$: Γ_{1+1} **extends** Γ_i

$\forall\, i \in \{1, \ldots, n\text{-}1\}$: $H(i) = \Gamma_i$

- Source method: *onStartCommand(Intent, int, int)* or *onHandleIntent(Intent)* callbacks
- Sink method: *startForeground(int, Notification)*

2. FOREGROUND SERVICES

ANALYSIS | DETECTION

RESULTS

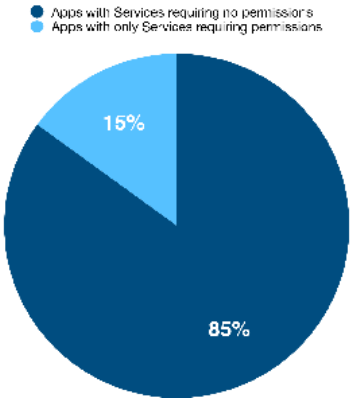
- Analyzed both third party apps and system apps
- Vulnerability was detected in at least one service in *every* application exposing a started service
- Few apps protect services with permissions
- System applications failures have serious consequences

2. FOREGROUND SERVICES

ANALYSIS DETECTION

RESULTS: THIRD PARTY APPS

- Top 20 most popular free applications in Google Play store
- Only 3 apps protect all their services with SIGNATURE permissions
- Of the 103 services analysed, only 3 were labelled (and confirmed) safe

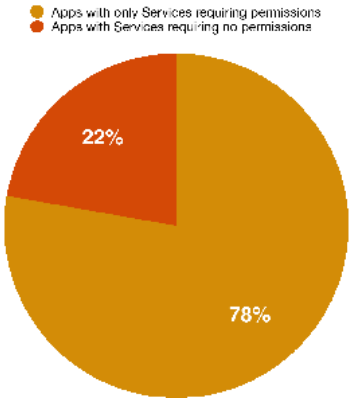


2. FOREGROUND SERVICES

ANALYSIS DETECTION

RESULTS: SYSTEM APPS

- Top 10 most popular system apps
- *Camera* exposes no services
- Only *Contacts* and *Settings* protect all services with DANGEROUS or SIGNATURE permissions
- Of the 64 services analysed, none were found to be safe
- 60 true positives, 4 false positives due to separate process in *Google Music*



3. UNEXPECTED INTENTS

APPROACH

- Assess readiness of exported components to be called by other apps
- Check exported services and activities in 10 system apps

RESULTS

- System apps *Settings* and *Phone* each expose a vulnerable activity
- Missing object initialization causes *NullPointerException* in called activity
- *Settings* and *Phone* will crash as a result
- An easy exploit?

3. UNEXPECTED INTENTS
APPROACH RESULTS

PHONE APP's CONSEQUENCES

- OS will restart *Phone* app automatically after crash
- Repeated crashes and restarts will lead to overall device crash and reboot
- Denial-of-Service attacks possible
- Google fix for Marshmallow and Oreo (August 2018)

4. EXPLOITING FOREGROUND SERVICES

USE EXPLICIT INTENT

- *Context* of service app
- *Class* object defining service

EXPLICIT INTENTS

- How to retrieve the Context object? Method *createPackageContext(String, int)*
- How to retrieve the Class object? Static method *Class.forName(String)*

EXPLOITING CLASSLOADERS

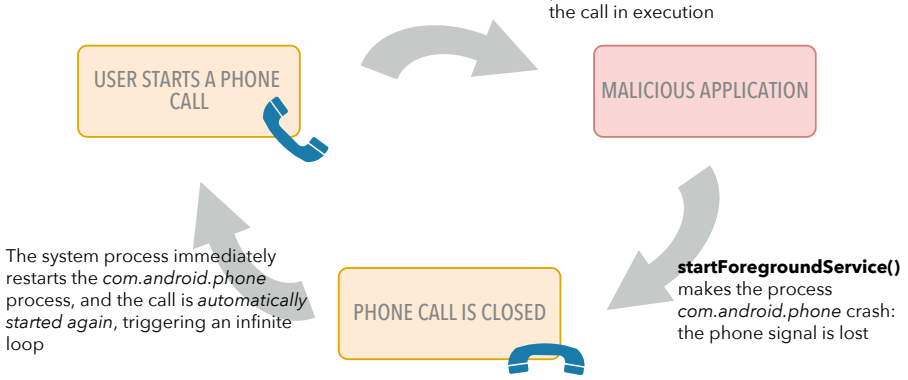
- Use Java *PathClassLoader* to load class object of OS apps and service app
- Use class loader and Java reflection API to:

1. Load class object that defines target service
2. Load *ContextImpl* OS class object holding context of malicious app
3. Modify package name and API level of malicious app in OS

4. EXPLOITING FOREGROUND SERVICES

THE PROBLEM CLASSLOADERS ATTACK EXAMPLE

ATTACK EXAMPLE: PHONE APP



DoS attack example targeting the Phone application.

5. CONCLUSIONS

5. CONCLUSIONS

CHANGE IN ANDROID PIE

FOREGROUND_SERVICE — added in API level 28

```
public static final String FOREGROUND_SERVICE
```

Allows a regular application to use `Service.startForeground`.

Protection level: normal

Constant Value: "android.permission.FOREGROUND_SERVICE"

Source: https://developer.android.com/reference/android/Manifest.permission.html#FOREGROUND_SERVICE

5. CONCLUSIONS

CONCLUSIONS

1. Introduction of *startForegroundService()* makes most apps susceptible to DoS attacks
2. Exported services and activities should be thoroughly tested against unexpected intents
3. Combination of Java class loader and Java reflection constructs makes system data structures accessible to malicious apps
4. No easy fix for (1) and (3) above

THANK YOU! QUESTIONS?