1. GOALS and MOTIVATION
1. INTRODUCTION

RESEARCH GOALS

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

MOTIVATION

Activity A

```
i = new Intent(...);
startActivity(i);
```

Activity B

```
_onCreate(...){
...
...
}
```

Activity A creates an implicit intent with the same values found in the intent filter declared by Activity B.

system process

Intent Filter

Action: x
Category: y
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 App:
```
i = new Intent(...);
startForegroundService(i);
```

Server App:
```
onStartCommand(...){
    ...
    startForeground(..., ...);
    ...
}
```
### 2. FOREGROUND SERVICES

#### ANALYSIS

#### DETECTION

### CLIENT AND SERVICE-SIDE COMBINATIONS

<table>
<thead>
<tr>
<th>CLIENT APP</th>
<th>SERVER APP</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>startService()</td>
<td>onStartCommand()</td>
<td>Background Service</td>
</tr>
<tr>
<td></td>
<td>startService() +</td>
<td>Foreground Service</td>
</tr>
<tr>
<td></td>
<td>onStartCommand() +</td>
<td></td>
</tr>
<tr>
<td></td>
<td>startForeground()</td>
<td></td>
</tr>
</tbody>
</table>
### CLIENT AND SERVICE-SIDE COMBINATIONS

<table>
<thead>
<tr>
<th>CLIENT APP</th>
<th>SERVER APP</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startService()</code></td>
<td><code>onStartCommand()</code></td>
<td>Background Service</td>
</tr>
<tr>
<td><code>startService()</code></td>
<td><code>onStartCommand()</code> + <code>startForeground()</code></td>
<td>Foreground Service</td>
</tr>
<tr>
<td><code>startForegroundService()</code></td>
<td><code>onStartCommand()</code> + <code>startForeground()</code></td>
<td>Foreground Service</td>
</tr>
</tbody>
</table>

### 2. FOREGROUND SERVICES

**ANALYSIS**

**DETECTION**

### CLIENT AND SERVICE-SIDE COMBINATIONS

<table>
<thead>
<tr>
<th>CLIENT APP</th>
<th>SERVER APP</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startService()</code></td>
<td><code>onStartCommand()</code></td>
<td>Background Service</td>
</tr>
<tr>
<td><code>startService()</code></td>
<td><code>onStartCommand()</code> + <code>startForeground()</code></td>
<td>Foreground Service</td>
</tr>
<tr>
<td><code>startForegroundService()</code></td>
<td><code>onStartCommand()</code> + <code>startForeground()</code></td>
<td>Foreground Service</td>
</tr>
<tr>
<td><code>startForegroundService()</code></td>
<td><code>onStartCommand()</code></td>
<td><strong>X</strong></td>
</tr>
</tbody>
</table>
2. FOREGROUND SERVICES

NEW DESIGN CHOICES

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

APPSEER: DETECTION TOOL

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

Reverse Engineering

File .apk

Jadx

ApkTool

Fetcher

ADB Intent Sender

Source-Sink search

Vulnerable components

java Sources

Exported components

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**

- **Hierarchy** $H$ of a service $\Sigma$:

  $$\begin{align*}
  H & \text{ is a set of classes: } H = \{ \Gamma_1, \Gamma_2, \ldots, \Gamma_n \} \\
  \Gamma_1 & \in \{ \text{"Service", "IntentService"} \} \\
  \Gamma_n & = \Sigma \\
  \forall \ i \in \{1, \ldots, n-1\}: \Gamma_{i+1} & \text{ extends } \Gamma_i \\
  \forall \ i \in \{1, \ldots, n-1\}: H(i) & = \Gamma_i
  \end{align*}$$

- **Source method**: `onStartCommand(Intent, int, int)` or `onHandleIntent(Intent)` callbacks

- **Sink method**: `startForeground(int, Notification)`

**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

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

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

<table>
<thead>
<tr>
<th>APPROACH</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Assess readiness of exported components to be called by other apps</td>
<td></td>
</tr>
<tr>
<td>• Check exported services and activities in 10 system apps</td>
<td></td>
</tr>
</tbody>
</table>
### 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?

### 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

THE PROBLEM

CLASSLOADERS

ATTACK EXAMPLE

USE EXPLICIT INTENT

- Context of service app
- Class object defining service
4. EXPLOITING FOREGROUND SERVICES

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

ATTACK EXAMPLE: PHONE APP

Thanks to the READ_PHONE_STATE permission, the attacker can detect the call in execution.

The system process immediately restarts the com.android.phone process, and the call is automatically started again, triggering an infinite loop.

5. CONCLUSIONS
5. CONCLUSIONS

CHANGE IN ANDROID PIE

<table>
<thead>
<tr>
<th>FOREGROUND_SERVICE</th>
<th>added in API level 28</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final String FOREGROUND_SERVICE</td>
<td></td>
</tr>
<tr>
<td>Allows a regular application to use Service.startForeground.</td>
<td></td>
</tr>
<tr>
<td>Protection level: normal</td>
<td></td>
</tr>
<tr>
<td>Constant Value: &quot;android.permission.FOREGROUND_SERVICE&quot;</td>
<td></td>
</tr>
</tbody>
</table>

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?