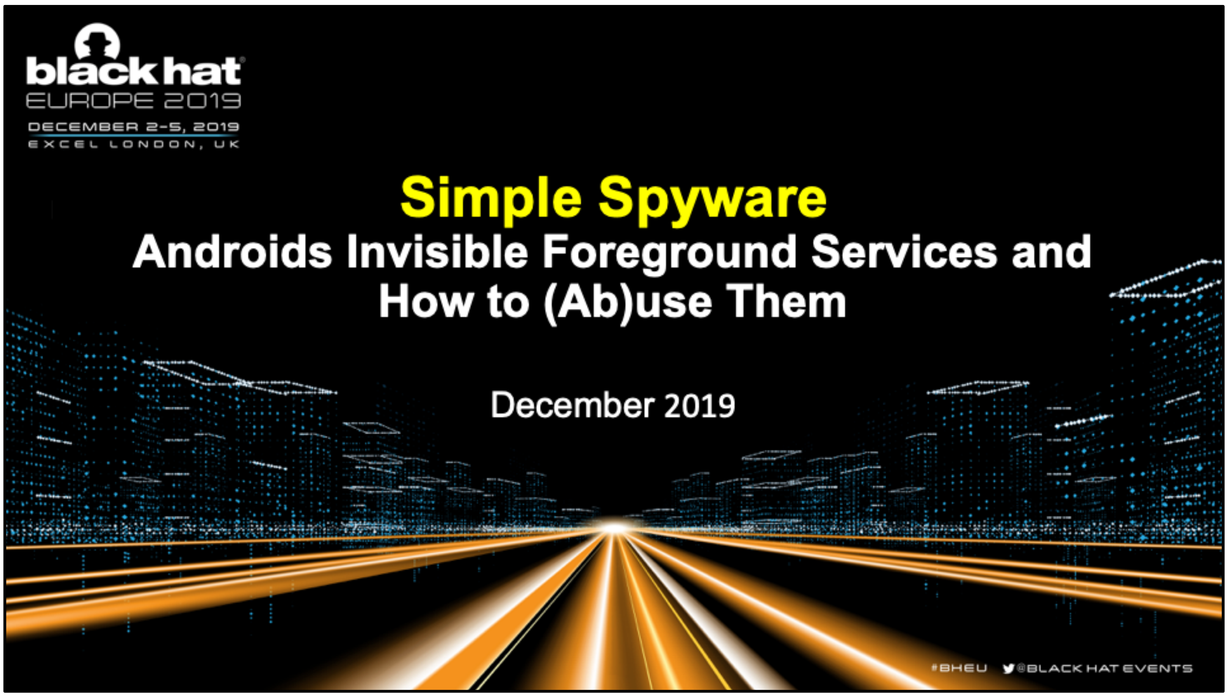


black hat EUROPE 2019

DECEMBER 2-5, 2019

EXCEL LONDON, UK

Simple Spyware

Androids Invisible Foreground Services and How to (Ab)use Them

December 2019

#BHEU @BLACK HAT EVENTS

Who am I

Thomas Sutter

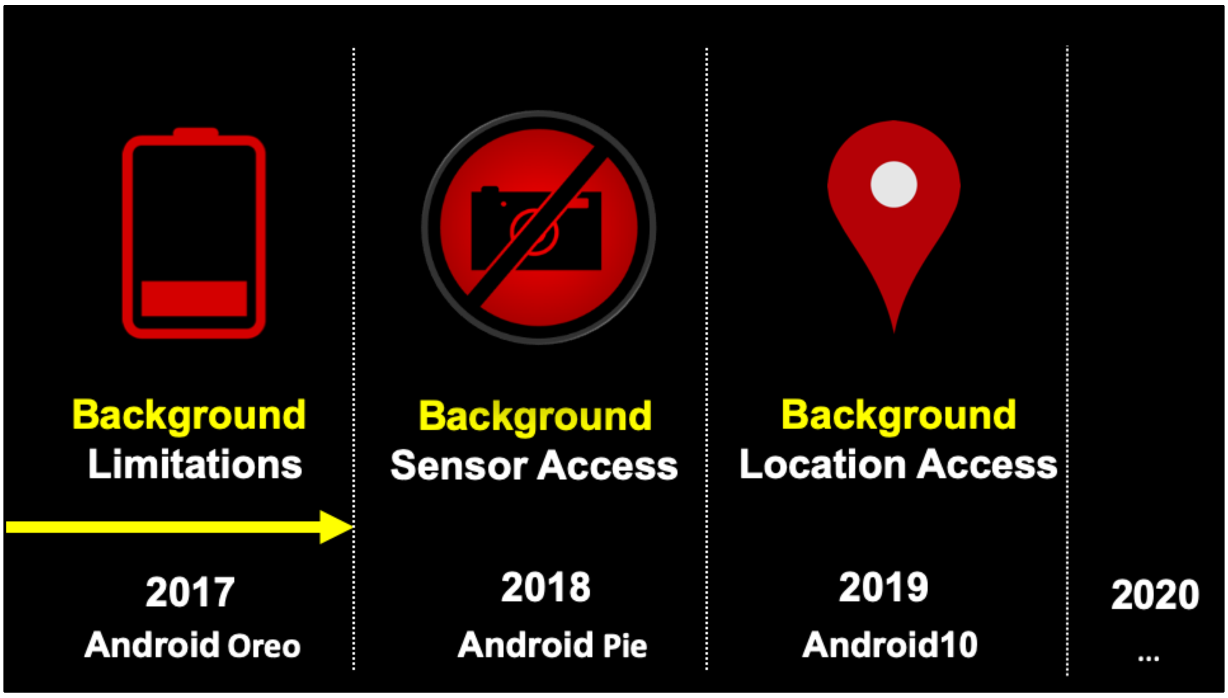
→ **ZHAW**: Research Assistant in Information Security @ Zurich University of Applied Sciences

→ **Student**: Master of Science in Engineering

→ **Contact**: suth@zhaw.ch or via Twitter @Me7e0r232

Short about me: I'm a research assistant at the Zurich University of Applied Science.
I work 50% as research assistant within the information security team and I'm doing a Master of Science in Engineering 50%.
The work presented within these slides is part of my first thesis for the MSc.

Let's start with the latest privacy changes



2017: Android starts to realize that access to sensor from Background is troublesome and that Apps should not run endlessly.
2018: The Access to Camera, Audio and other sensors from background gets restricted/blocked.
2019: Android introduces a new Permission Level for Locations. “Allow only while in use”.

Conclusion: Security is increasing with every version. Background capabilities seems to get more and more limited

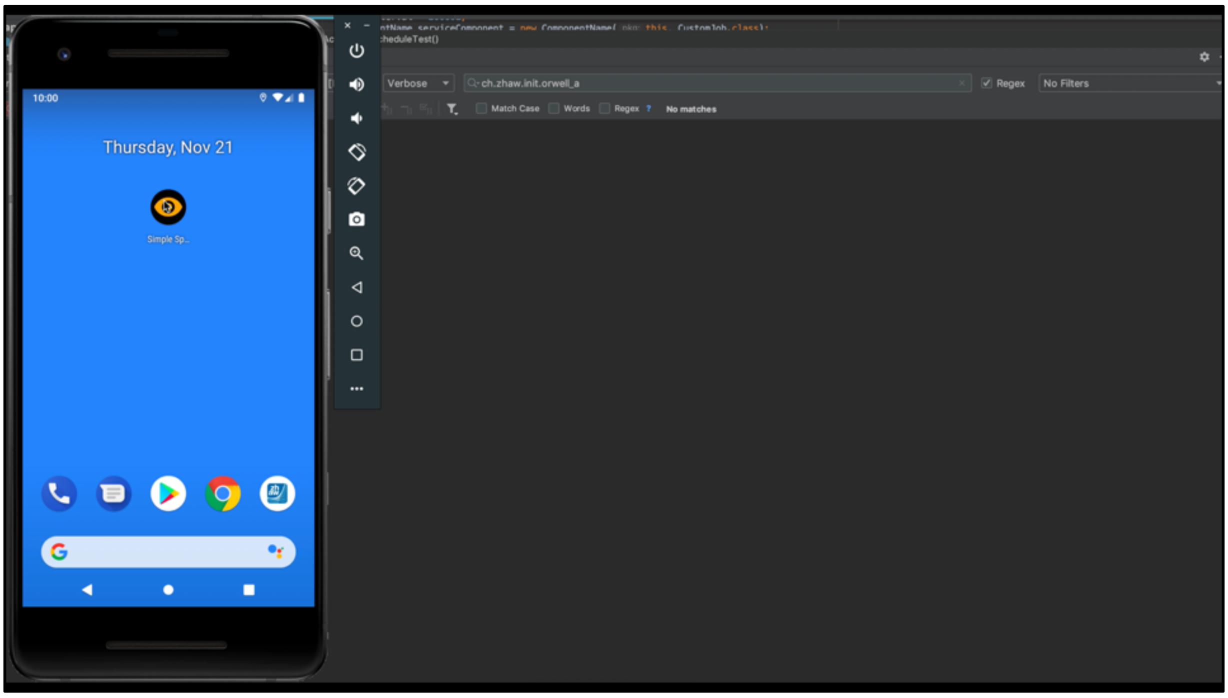
LIVE DEMO

Within the live demo we show that the access to the camera from “background“ is possible without any visual sign.

But that's nothing new!
So what's new?

Spying on sensors like the camera is nothing new. Several researchers have shown bugs/attacks doing the same thing.

What is new in this case is how we access the sensors. The example shows does not need heavy exploitation, we just use Androids standard API calls.



```
ntName serviceComponent = new ComponentName( pkg: this, CustomJob.class);
heduleTest()
```

Verbose | ch.zhaw.init.orwell_a | ✓ Regex | No Filters

Match Case | Words | Regex | No matches

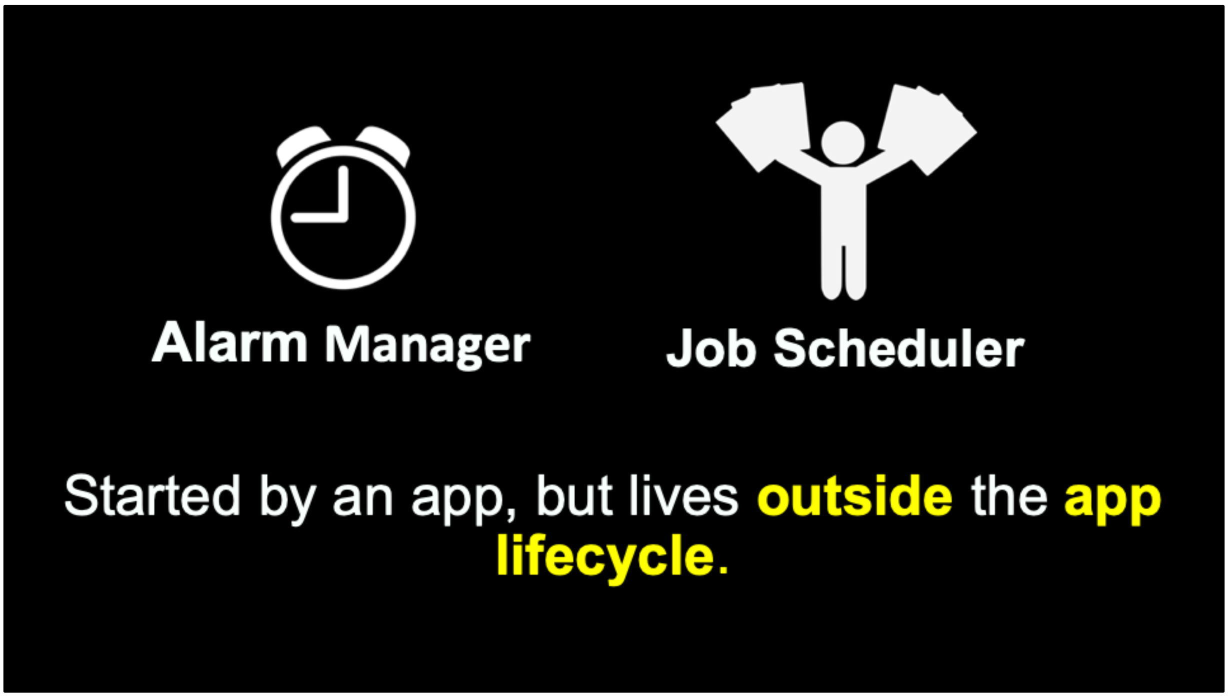
Shows a video of what has change with Android Oreo and Pie when it comes to Background Service execution.
https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)

The video shows that background services can’t be started from a background context.
The OS throws an IllegalStateException and does not start the service, hence the we can’t access the sensors of the phone.

First step
How to run stuff in Background?

Schedulers

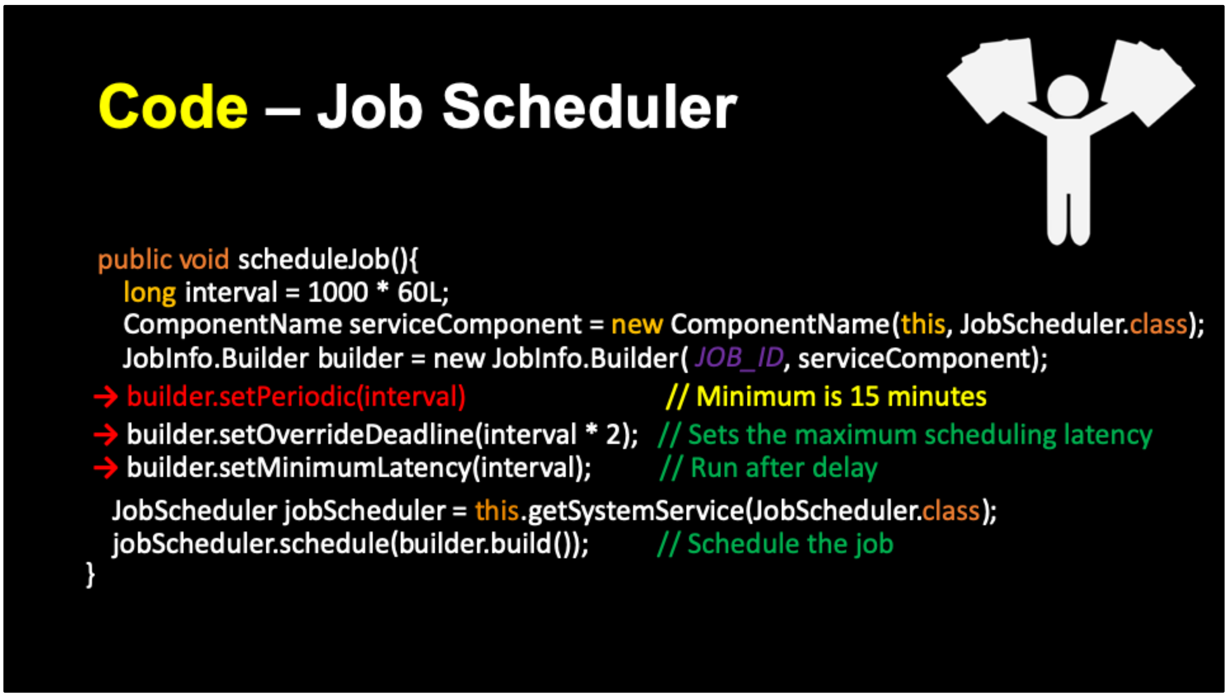
After an app is installed, we need to find a way to persist the app and to execute code in the background. The first challenge to overcome is to run permanently in the background in order to spy on as much data as possible. The Android API offers an API for scheduling tasks in background.


Alarm Manager

Job Scheduler

Started by an app, but lives **outside** the **app lifecycle**.

Alarm Manager:
https://developer.android.com/reference/android/app/AlarmManager
JobScheduler: https://developer.android.com/reference/android/app/job/package-summary
There are several schedulers within the API:
https://developer.android.com/guide/background/#choosing_the_right_solution_for_your_work
We focus on Job Scheduler rather than Alarm Manager. Alarm Manager has been widely used for Malware in recent years and can be used as well for the demonstration.
Schedulers in general get started from the app's context and then live partly outside the app lifecycle. That means, when a user closes the app, the scheduler survives and can execute code in the background.
The operating system takes care of prioritizing tasks. Depending on the vendor different task optimization approaches are used.
Schedulers are used for all kinds of apps. Often apps use it in order to synchronize data or process data in background.



Code – Job Scheduler

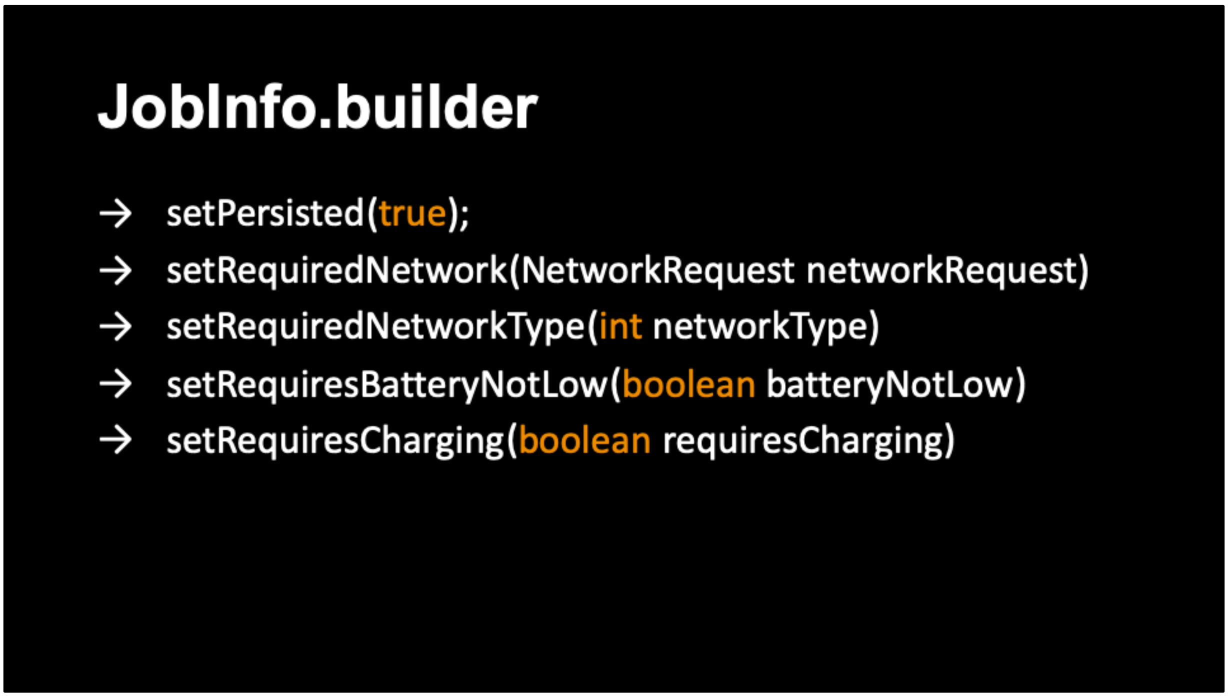
```
public void scheduleJob(){
   long interval = 1000 * 60L;
   ComponentName serviceComponent = new ComponentName(this, JobScheduler.class);
   JobInfo.Builder builder = new JobInfo.Builder( JOB_ID, serviceComponent);
→ builder.setPeriodic(interval)                    // Minimum is 15 minutes
→ builder.setOverrideDeadline(interval * 2);  // Sets the maximum scheduling latency
→ builder.setMinimumLatency(interval);          // Run after delay
   JobScheduler jobScheduler = this.getSystemService(JobScheduler.class);
   jobScheduler.schedule(builder.build());         // Schedule the job
}
```

We focus on JobScheduler, cause it is a good example on how to circumvent limitation within Androids API. A task with Job Scheduler can be created by the JobInfo.Builder class. It allows to set some options for a scheduling strategy, which can be used for spyware as well.

When we want to run a periodic job we can use the .setPeriodic(interval) method, but this method is limited. You can't set an interval under 15 minutes, cause the systems defines this minimum. So in order to use this API for spyware, which we want to run in a shorter period of time, we need to get around this limitation. To do so, we can use the API's delay function "setMinimumLatency(interval)". With the delay function we can build our own periodic job, since there is no limitation on how many jobs we are allowed to spawn in which time. The idea is as follows:

- We start a job with setMinimumLatency(interval).
- When the job is executed we start a new job with a delay within it's execution.
- We repeat this steps as long as necessary

What we get is a periodic job that can execute every couple of seconds. The operating system may reschedule the job depending on the phones state, but it gets executed at some point.


JobInfo.builder

→ setPersisted(true);
→ setRequiredNetwork(NetworkRequest networkRequest)
→ setRequiredNetworkType(int networkType)
→ setRequiresBatteryNotLow(boolean batteryNotLow)
→ setRequiresCharging(boolean requiresCharging)

Some details about the JobScheduler API:

If you close the app it still runs until the operating system stops them.
Job Scheduler and Alarm Manager in general:

- Are in the background and should do not have the right to access sensors, since it's a background context.
- Live outside of the app lifecycle
- Can run when the device is in stand-by and survive restarts
- Scheduling new tasks within a scheduler is possible
- Closing the app does not stop schedulers, only force stop does.
- Are allowed to start foreground services.

Limitations:

Have no direct access to critical system sensors
Normally, JobScheduler can only run every 15 minutes
Use setMinimumLatency() and schedule a new Job within the execution of the delay
Allows to schedule under 15 minutes
The system can reschedule tasks if it needs the smartphones resources.
If the phone is for a longer period of time in stand-by the task is likely to get

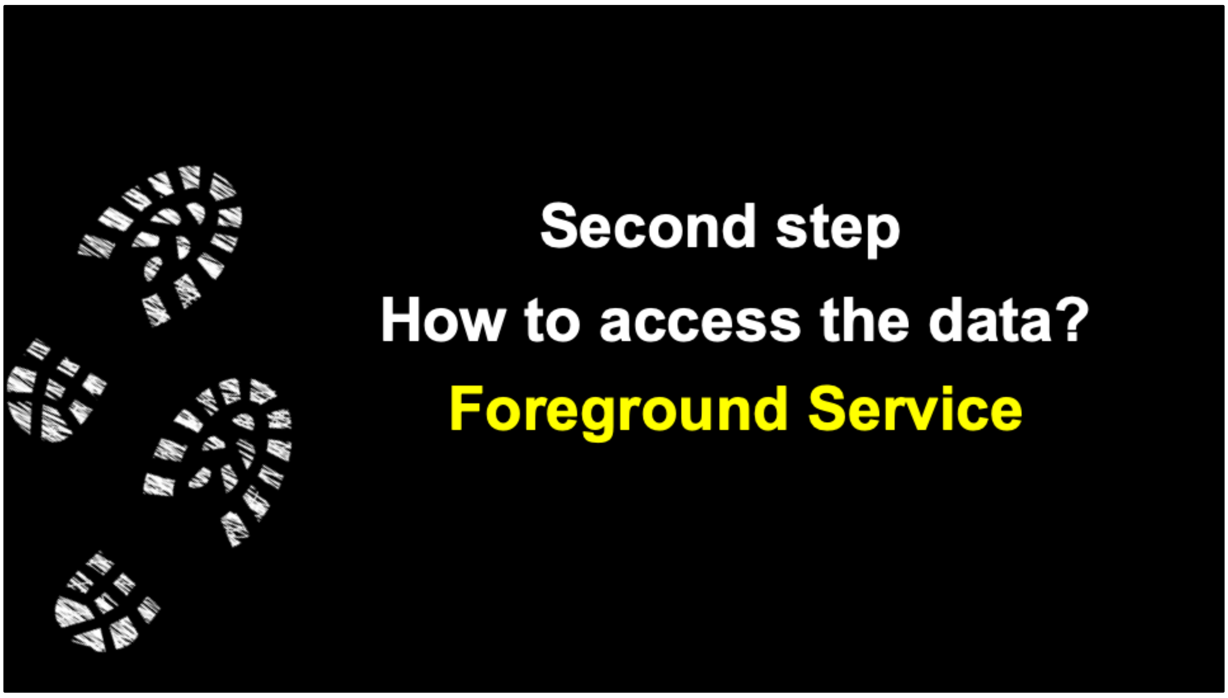
rescheduled. Some vendors may kill the task after some days in case the app is never used.

For Malware:
setPerstisted(true): Persist across device reboots. Needs a permissions (RECEIVE_BOOT_COMPLETED)
setRequiredNetwork() and setRequiredNetworkType(): Set the job to run only if the phone is connected to a specific network.

- An attacker can use this for example to make an analysis of network traffic harder. For example: Only sync when the phone is not connected to wifi
- Upload large files only when phone is connected to wifi, so that the user does not get billed for mobile traffic.

setRequiresBatteryNotLow and setRequiresCharging():

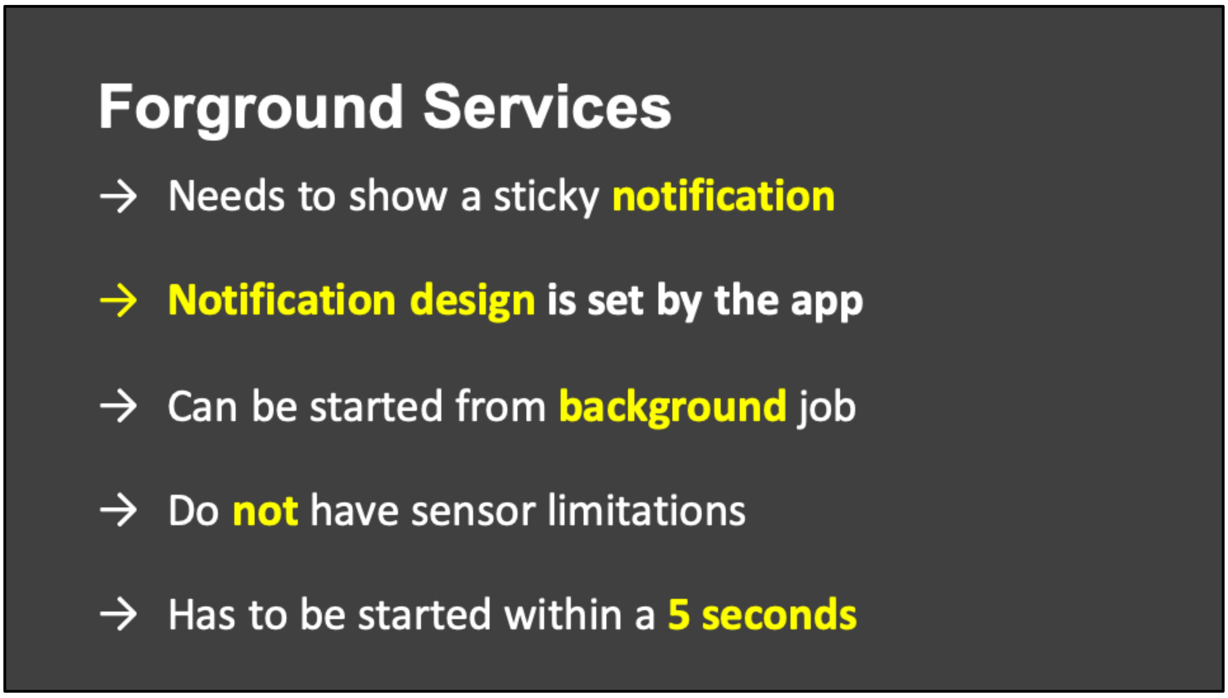
- Upload Files only when connected to wifi and the battery is charging. The user might not watch his phone activity during charging,
- People tend to charge their phones over night. So ideal moment to upload data unnoticed.



Second step
How to access the data?
Foreground Service

Now we've got something running in Background how do we get the access to sensors?
JobScheduler is a background task and has therefore no access to critical sensors. As we saw in the first video, a normal background service can't be started from a JobScheduler, but a foreground service can.

What's the difference of fore- and background services anyways?



Forground Services

→ Needs to show a sticky **notification**

→ **Notification design is set by the app**

→ Can be started from **background** job

→ Do **not** have sensor limitations

→ Has to be started within a **5 seconds**

Overview of Android services:
https://developer.android.com/guide/components/services

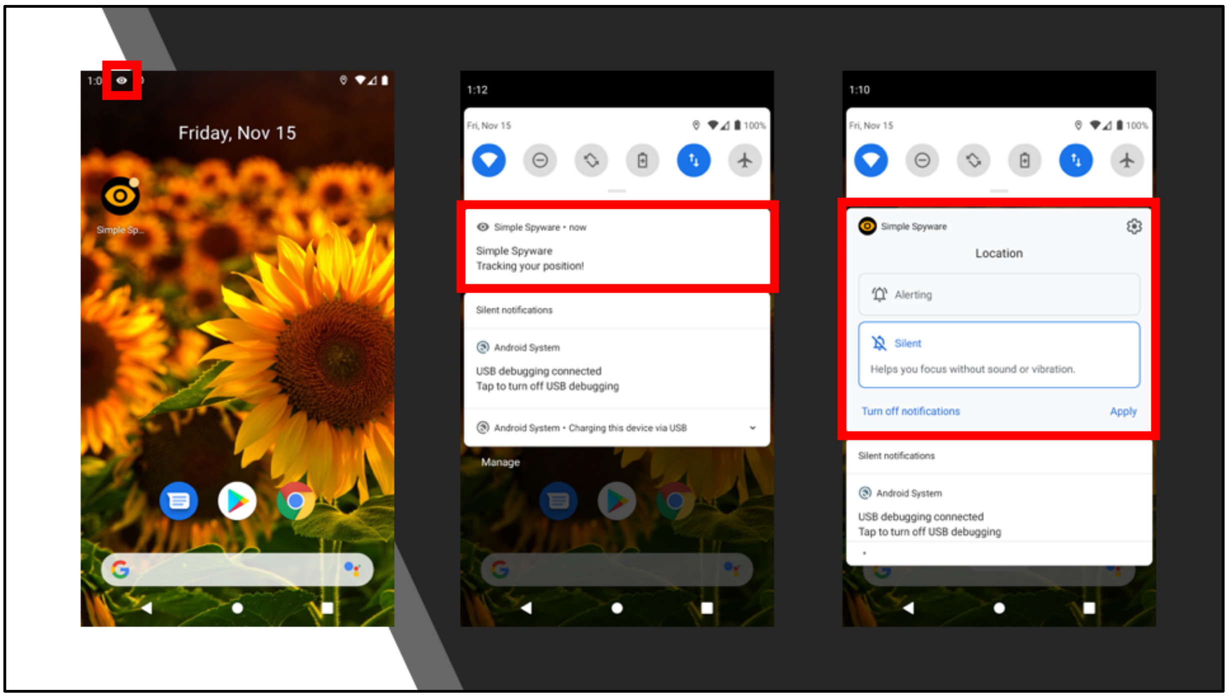
Foreground services do not have limitations in Android since they are visible to the user. That means:

- Can track the precise location within a short interval. You can get an infinite number of location per hour. Usually there is a limitation «Background location gathering is throttled and location is computed, and delivered only a few times an hour.» Have a look at: https://developer.android.com/guide/topics/location/battery for more information.
- Can access critical sensors like camera, microphone, proximity sensor etc.
- Foreground services are visible to the user in form of a sticky notification. This means, that the notification can't be swiped away by the user. The user only can disable the notification channel in order to not show the notification.
- The design of the notification can be set by the app developer.
- In order to start a foreground service an app must use the startForeground Method: https://developer.android.com/reference/android/content/Context.html#startFore

groundService(android.content.Intent)
  - We have 5 seconds to start this method, otherwise the app crashes with ANR.

Background services are limited to use within the app. So if the app gets closed a background service is stopped after some minutes.

- Location Tracking over background services is limited to few times an hour
- Critical sensor access to camera and microphone is denied
- Execution can be stopped by the operating system at any time



Simple Spyware • now
Simple Spyware
Tracking your position!

Silent notifications

Android System
USB debugging connected
Tap to turn off USB debugging

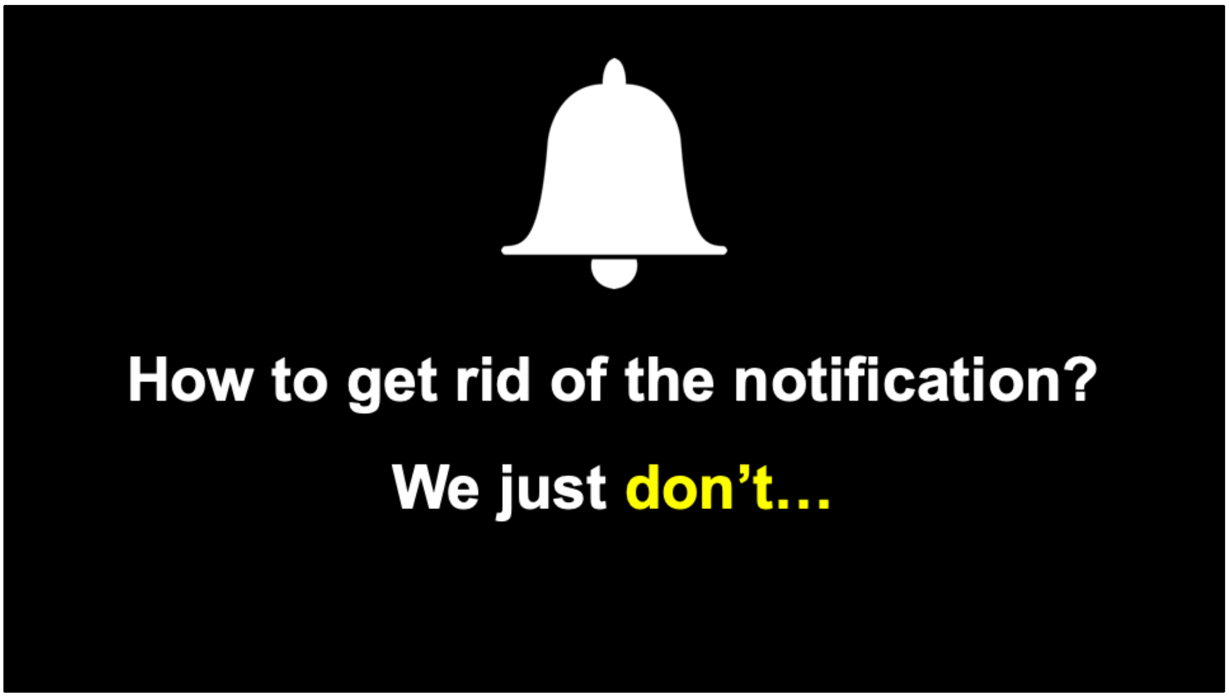
Android System • Charging this device via USB

Simple Spyware
Location
Alerting
Silent
Helps you focus without sound or vibration.
Turn off notifications
Apply

Silent notifications

Android System
USB debugging connected
Tap to turn off USB debugging

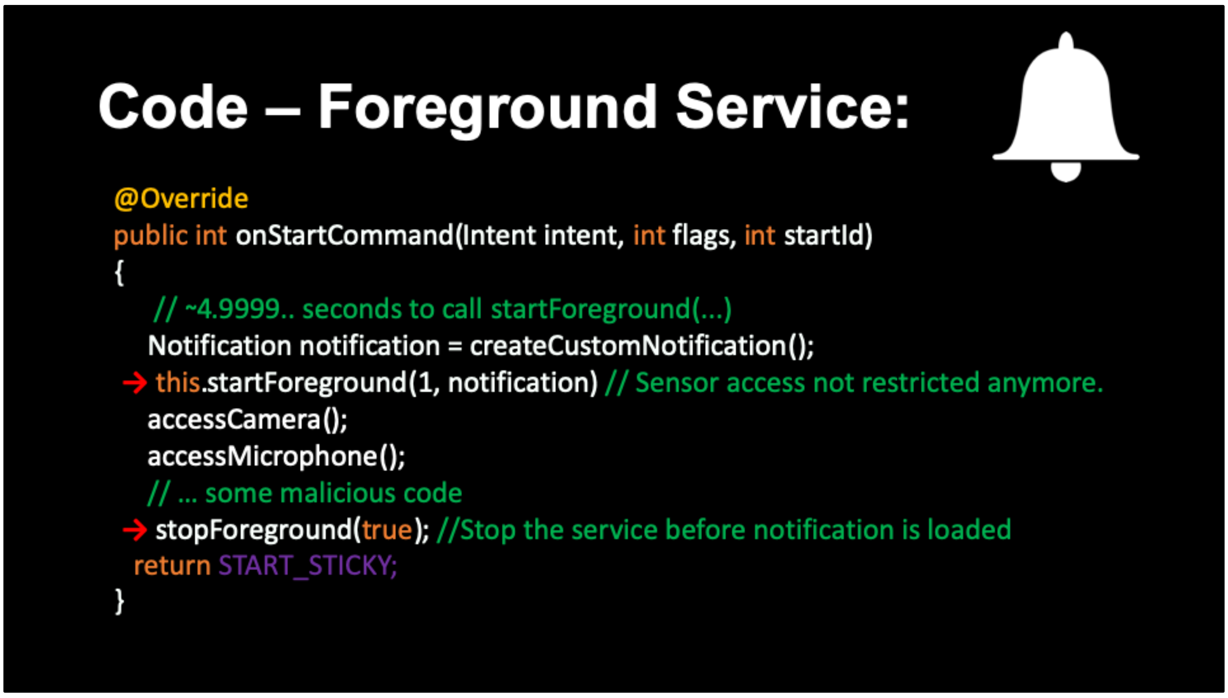
Example of a Foreground Service:
On the first picture you see that foreground services are usually shown in the top left corner of the screen. The symbol shown can be set by the app, so it is possible to set unsuspicious icons.
On the second picture you see that a foreground service shows a sticky notification, which can’t be swiped away by the user.
On the third picture you see that a user can turn off notification of the service if he wants to.



How to get rid of the notification?

We just **don't…**

A permanent notification would raise the user's attention sooner or later. In order to produce effective spyware an attacker can either set an unsucspicious icon or get rid of notification somehow.

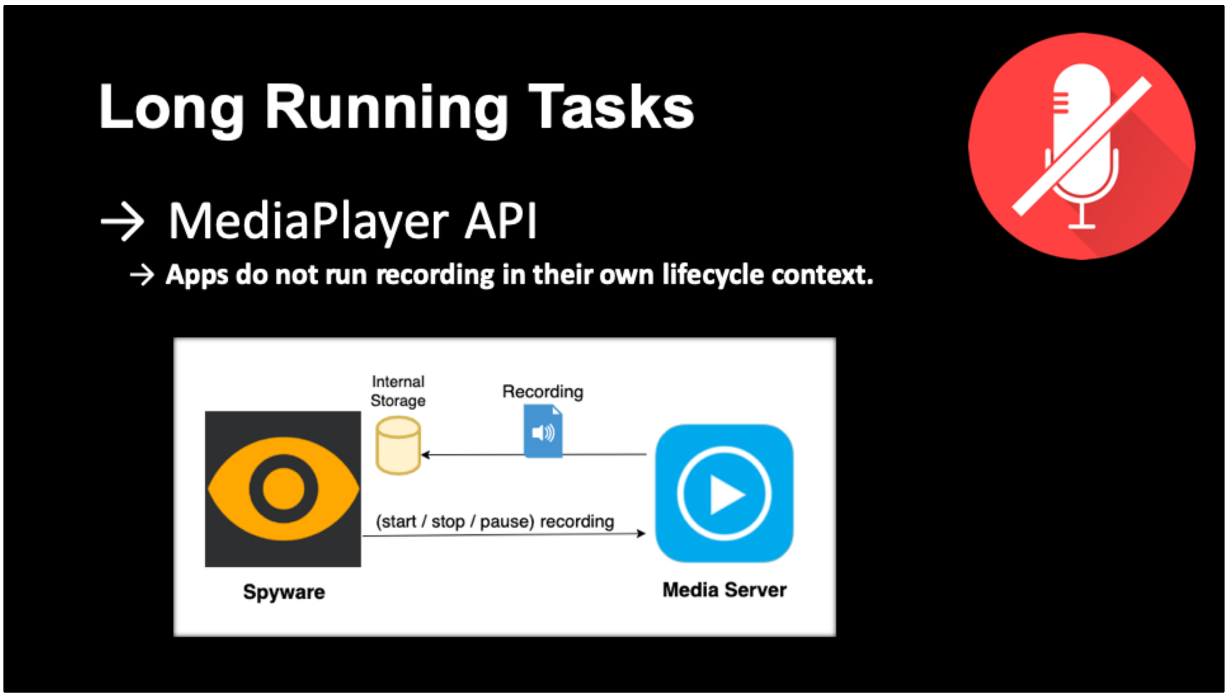

Code – Foreground Service:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
    // ~4.9999.. seconds to call startForeground(...)
    Notification notification = createCustomNotification();
 → this.startForeground(1, notification) // Sensor access not restricted anymore.
    accessCamera();
    accessMicrophone();
    // ... some malicious code
 → stopForeground(true); //Stop the service before notification is loaded
  return START_STICKY;
}
```

This code snippet shows the onStartCommand method of a foreground service. As mentioned before, a foreground service needs to call the startForeground method within a five second time window.
What we see in this code snippet is a race condition: As soon as startForeground is called the access to critical sensors is not restricted anymore and parallel the operating system starts to load the notification.

The notification needs some seconds to show up (usually between 2 and 5 seconds). So an attacker can simply access a critical sensor and then directly kill the foreground service again. Taking a picture from the camera takes around one second, so we can actually access the sensor before the notification shows up.

This race condition combined with a job scheduler allows to spawn jobs, which can access any sensor, without any visible indicator. (The limitation of this approach is discussed later)



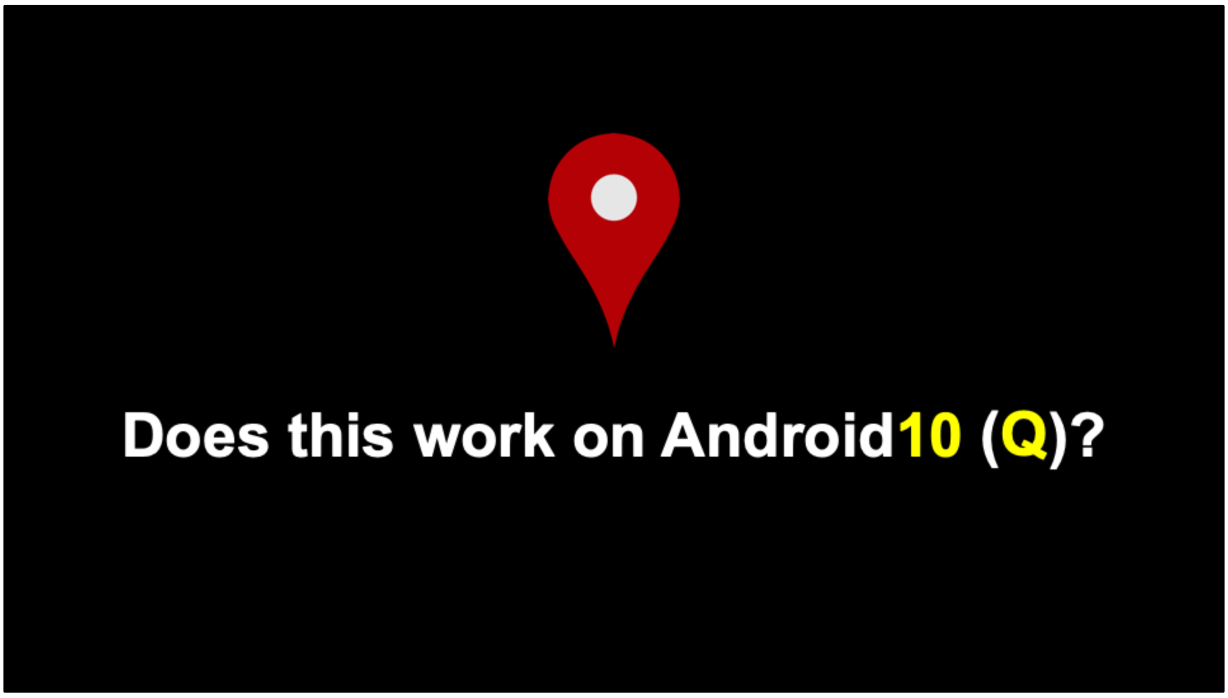
Long Running Tasks

→ MediaPlayer API

→ Apps do not run recording in their own lifecycle context.

The approach uses short lived foreground services. If we want to do a long running task like recording audio we could start the audio recording and stop it immediately again as shown before with invisible foreground services. We could do this in a high frequency like every second in order to record over a longer period of time. This would of course generate a lot of audio files and we would probably miss some of the audio we want to record. So instead we use the Media Player API which takes care of this problem for us.

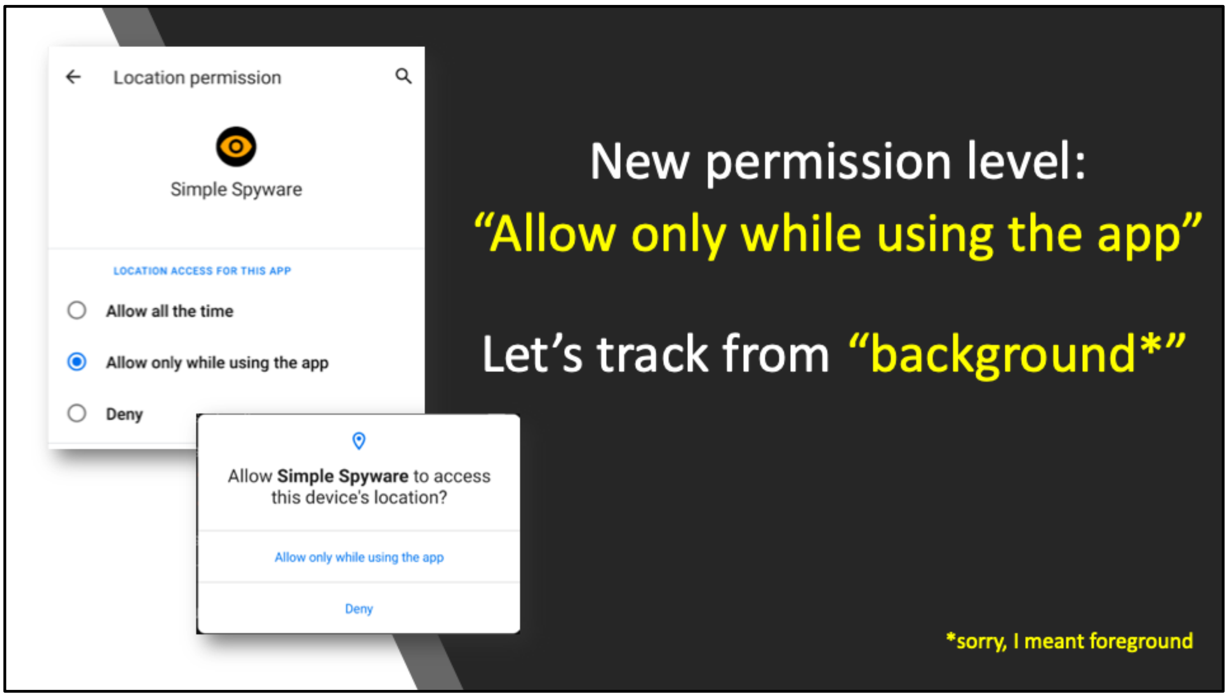
Since Androids Media Player API takes care of audio recording all we have to do within the spyware is to start the recording. The media player will then execute the recording within its own context. That means even when our job is finished the media player will still record the audio. All we have to do is simply stop the recording after some minutes or hours and the media player will save the recording within an audio file. Depending on the configuration the audio file then can be saved within the apps internal storage, so that a normal user will never have access to the recordings.



Does this work on Android10 (Q)?

All the shown approaches seem to work as well on Android10 devices. Since I do not own an Android10 device so far I tested this only on the Android10 Emulator and a friends Pixel phone. Other phones may behave differently.

Remember slide number four. There where privacy changes to location tracking in Android10.


New permission level:

"Allow only while using the app"

Let's track from "background*"

*sorry, I meant foreground

Location permission

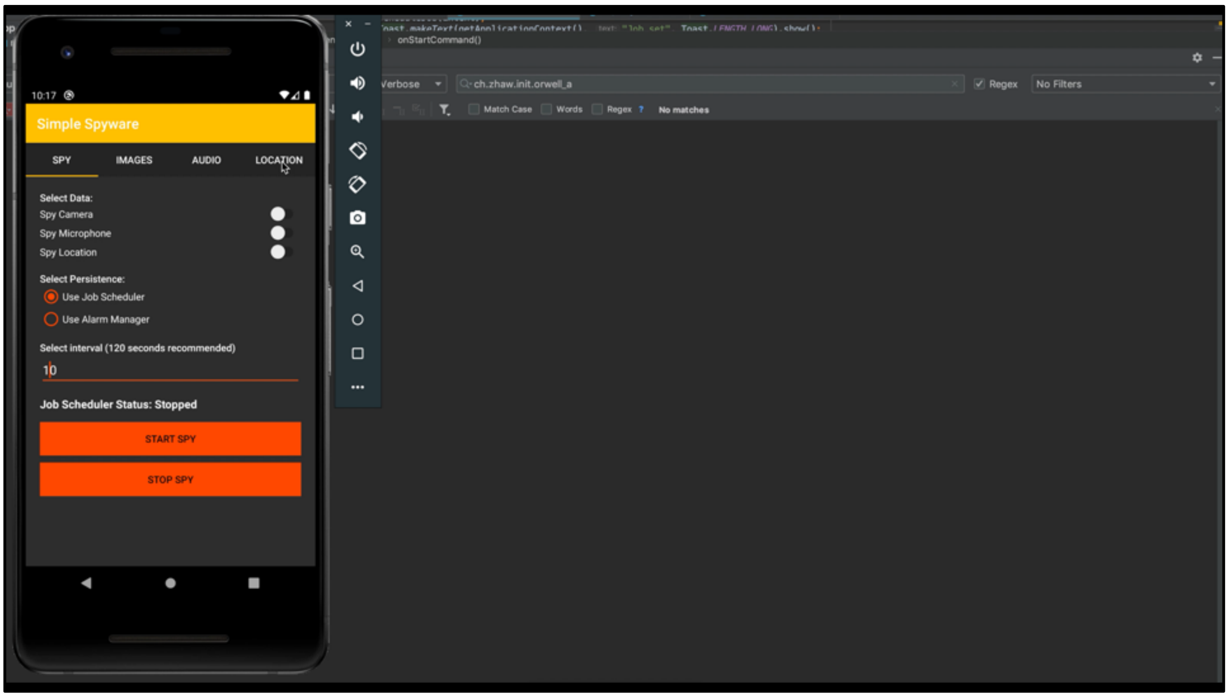
Simple Spyware

LOCATION ACCESS FOR THIS APP

Allow all the time [✓ Allow only while using the app] Deny

Allow **Simple Spyware** to access this device's location?

Android10 introduces a new permission level for location tracking. The "Allow only while using the app" level should allow apps to only track their users when the app is actively used. Foreground service counts as using the app, therefore the access is granted and we can track the user even when the app is closed.



Select Data:
Spy Camera
Spy Microphone
Spy Location

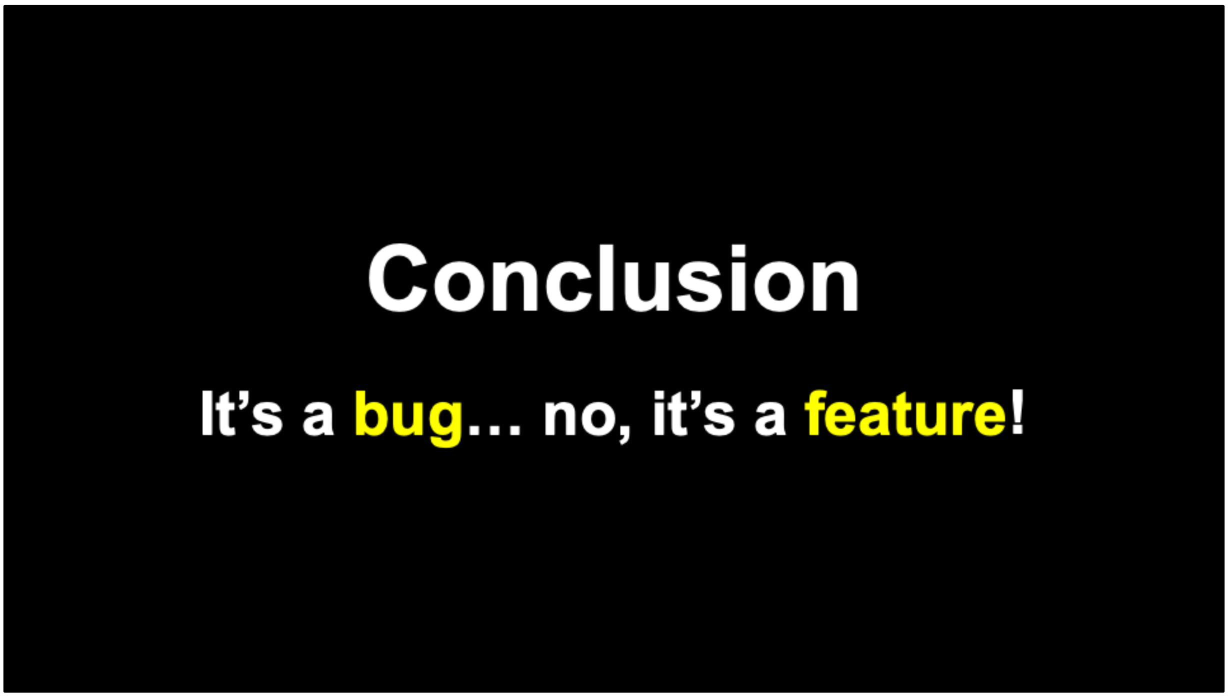
Select Persistence:
[✓ Use Job Scheduler]
Use Alarm Manager

Select interval (120 seconds recommended)
10

Job Scheduler Status: Stopped

Since we are indoors, I can only show a Video of an Android10 emulator with simulated GPS tracking. The video shows that tracking is possible without the foreground services notification showing even when the „Allow only while in use“ permission is set.

If you have a close look at the top menu bar, you can see that the location notification pops up and directly disappears again. This behavior is vendor specific. Some phones show the location icon only when an app is accessing the location and other phones show it permanently when the GPS is activated. So tracking on some vendors work better than others.

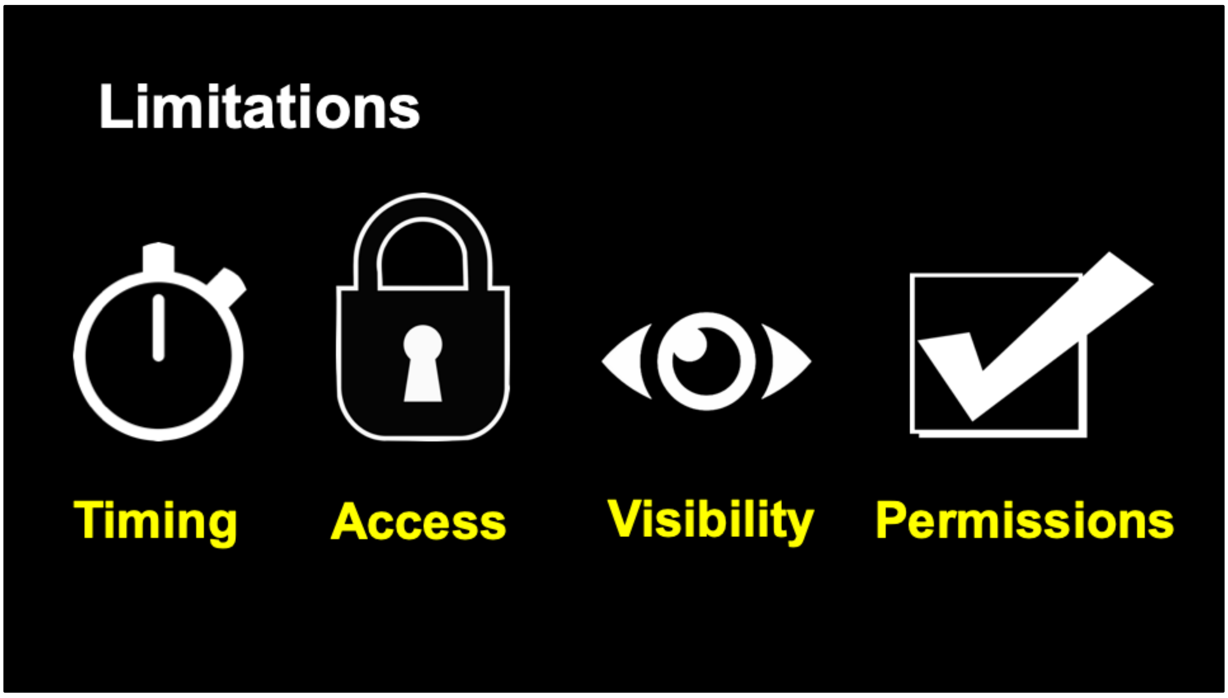


Conclusion

It's a **bug**… no, it's a **feature**!

Conclusions:

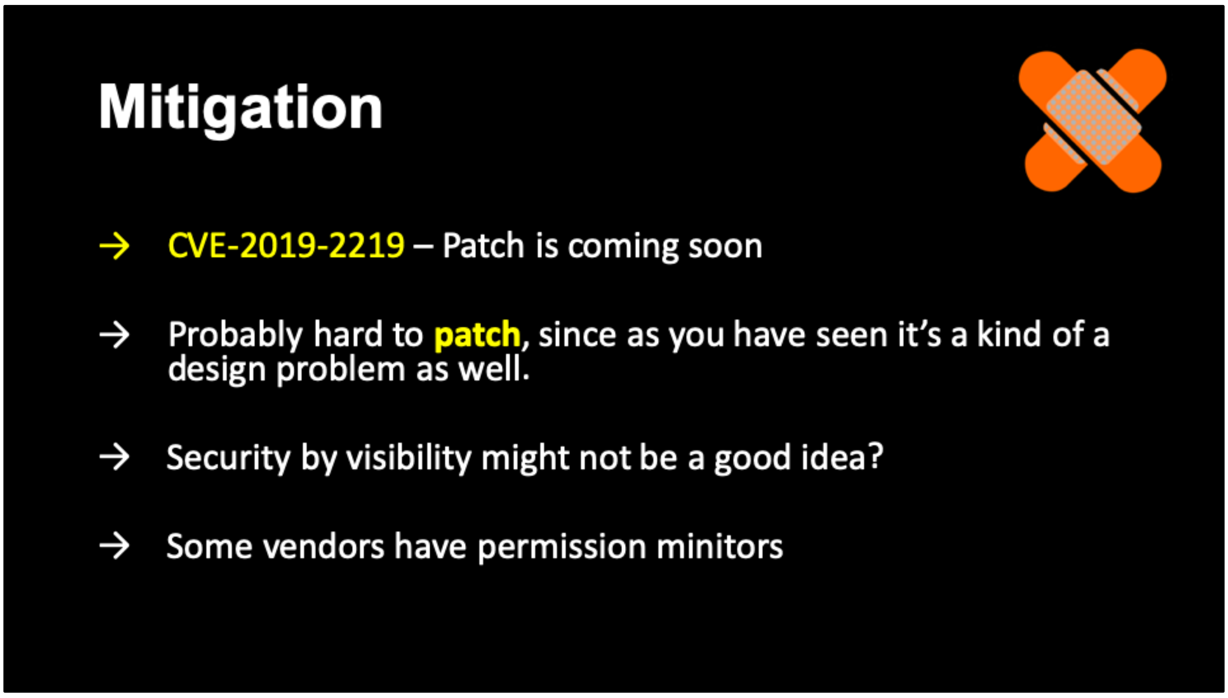
- Android security is increasing, but there are some bugs within the architecture. As shown we can get around some limitations without heavy exploitation.
- Spying on the users is simpler than it should be. I'm a student and I found this within two months of research, imagine what a team of attackers can do.
- Android lacks when it comes to transparency. The user has no idea which app is running in the background.
- Relying on the user interface for access to critical sensors might not be a good idea. Remember an attacker can set the design of the notification.



Limitations

Timing
Access
Visibility
Permissions

This approach has as well, some drawbacks/limitations:
Timing: Scheduling behavior varies with smartphone vendors. Execution times may change when the operating system needs the resources. There are as well vendors which stop jobs after some days when the app wasn't started by the user.
Access: Only one app at a time can access a sensor. If the spyware tries to access the microphone during a call either the phone app does not work or the spyware.
Visibility: Some vendors have implemented permission monitors, which can detect such abusive behavior. During testing, we found out that some vendors have preinstalled permission monitors, which warn the user of background activity. We noticed that only the location tracking was detected. Other sensors like the camera and microphone were not detected during testing.
Permissions: You need the permission to access a sensor. This approach does not include privilege escalation.
In practice it is hard to use this for mass surveillance, but single targets could be a problem. This issue will hopefully be patched soon, but it shows that spyware in general is a problem.
Most users think they are safe because this attack needs the users permission, but remember that permissions like the "Allow while only in use" may not behave like intended.

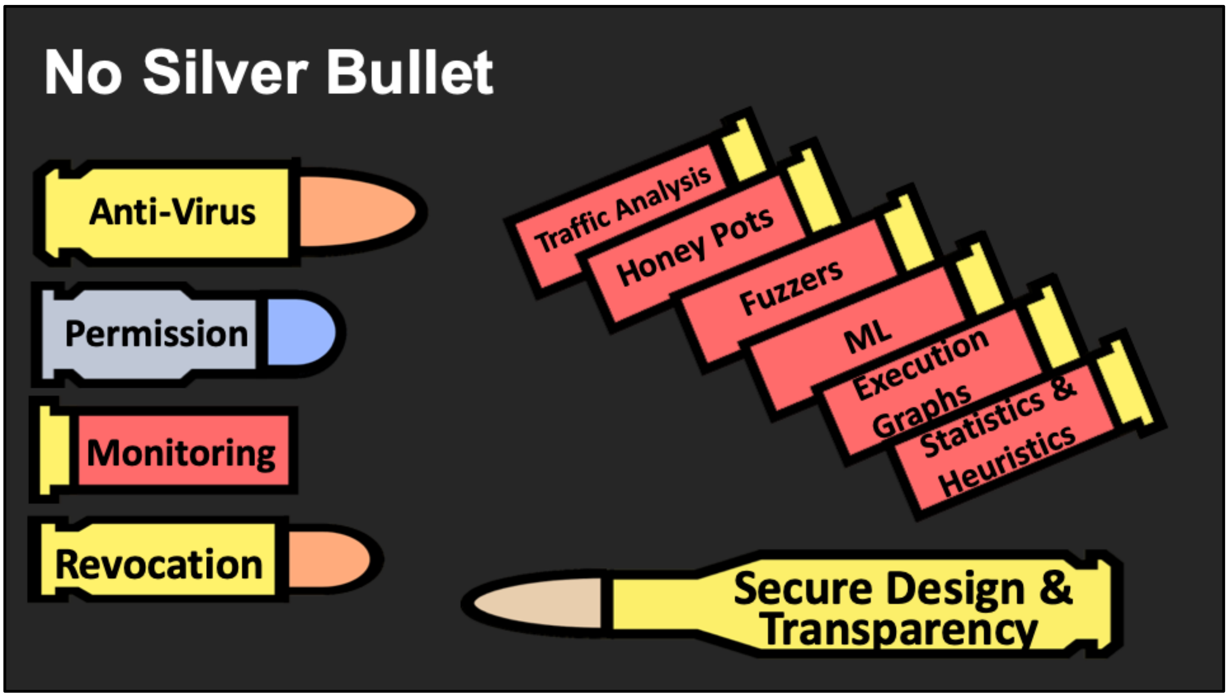


Mitigation

- → CVE-2019-2219 – Patch is coming soon
- → Probably hard to **patch**, since as you have seen it’s a kind of a design problem as well.
- → Security by visibility might not be a good idea?
- → Some vendors have permission minitors

This vulnerability will be patched soon.

If the security relies on the visibility of one notification it might be a good idea to rethink the concept of foreground services. For example: Why give the attacker the opportunity to show and design the notification, instead of giving the camera service the task to do so.



No Silver Bullet

Sadly, I cannot provide a silver bullet for spyware.

What users can do is on the left side of the images. There are several possibilities to make it harder for attackers to spy on the users. These are just some ideas.
On the right side are techniques which are used for Spyware detection.

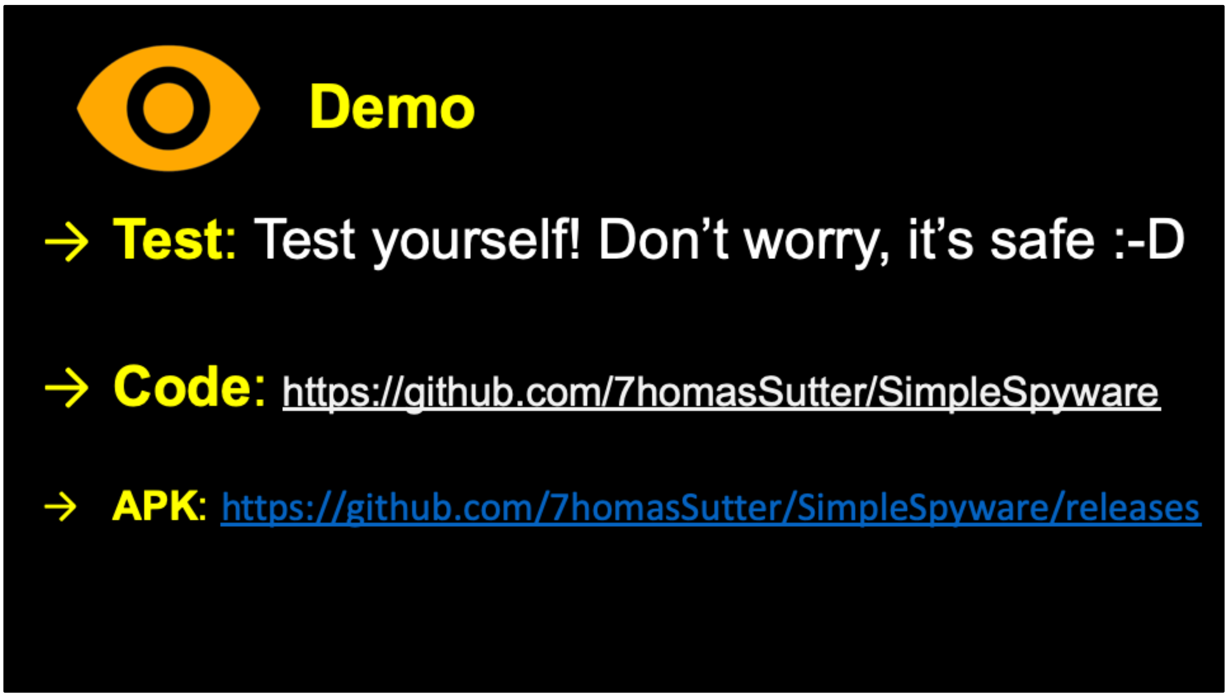
Takeaways

Stop Apps

Revoke Access

If you are unsure if an app is trustworthy, then better completely stop it when you don’t need it.

As long as Android has these problems I would advise users to force stop apps and revoke all access permissions after use:
Stop Apps: Force stop apps which do not need to run permanently. This stops the execution of Job Scheduler and Alarm Manager as well.
Revoke Permissions: After use, remove for every app their permissions. Most apps do not need permissions in the background.



Demo

→ **Test:** Test yourself! Don't worry, it's safe :-D

→ **Code:** https://github.com/7homasSutter/SimpleSpyware

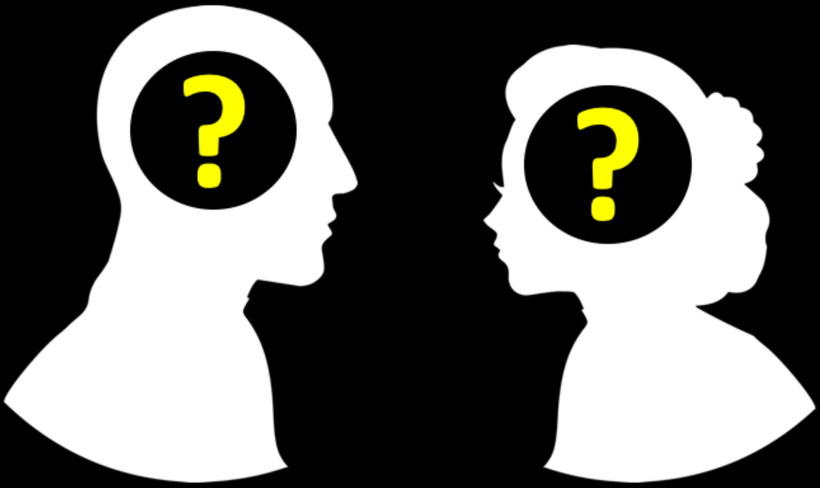
→ **APK:** https://github.com/7homasSutter/SimpleSpyware/releases

All your data stays on your phone. The demo is 100% offline. Have fun testing it and let me know what you think about this problem.

I hope you enjoyed this short talk…

Thank you!

there is some more info in the appendix
and the whitepaper



Contact: suth@zhaw.ch or via Twitter @Me7e0r232