When Security is not a Developer's fault.

Rodrigo Chiossi
r.chiossi@samsung.com

AndroidXRef: One year ago…

- Online source code cross reference of the Android source code.
- All major Android versions available.
- Average 10K page views per day.

www.androidxref.com

SIDI: Samsung's Research Lab

- Main Mobile Research Lab in Latin America.
- Focused on Smartphone research.
- Strong research on Mobile Security.
  - Focus on offensive security.

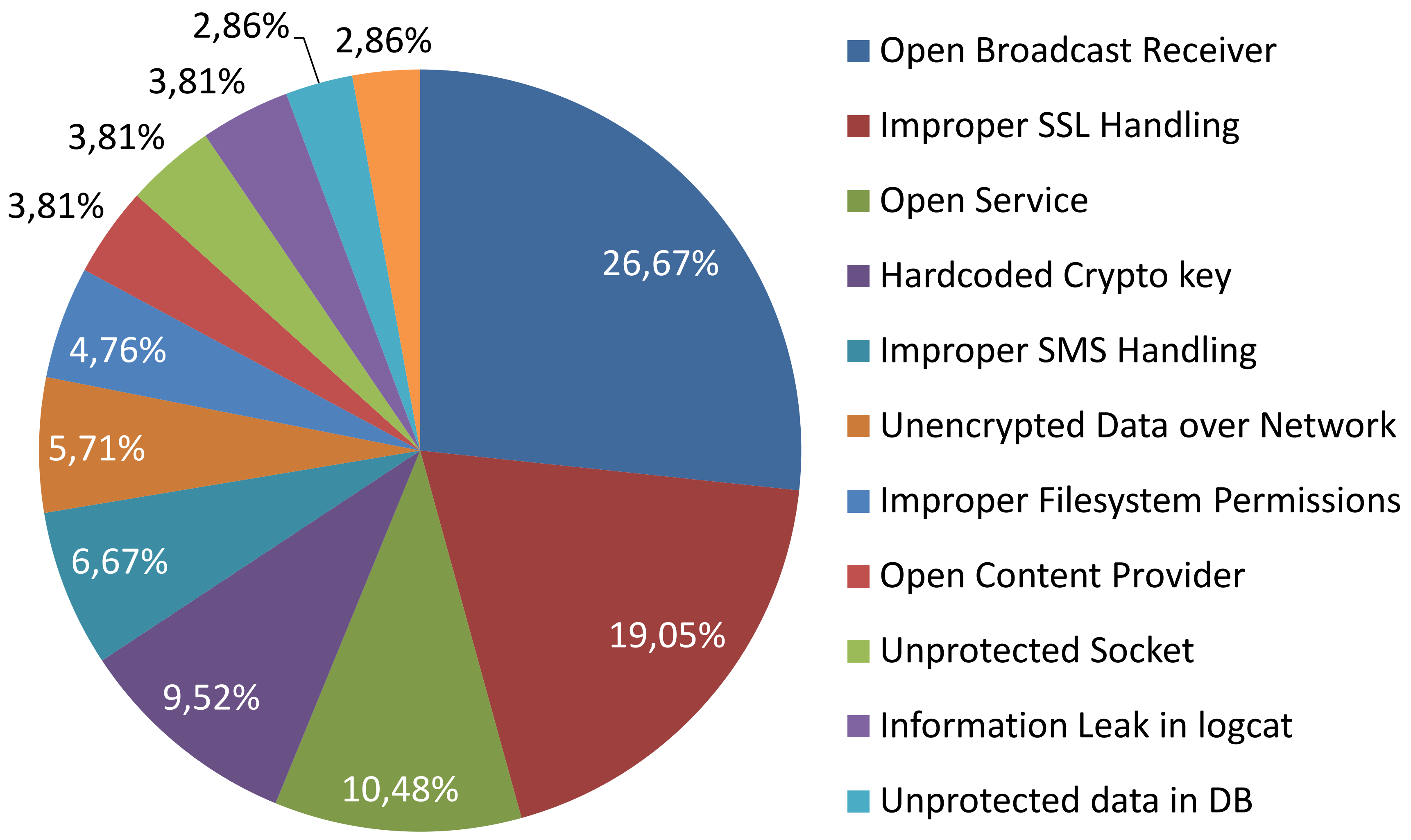
Security Targets

- Kernel
- File System
- Android Platform
- Android Applications

Apps Analyzed

- Pre-Loaded Apps
  - Samsung Apps
  - Partner Apps
- Non Pre-Loaded Apps
  - Samsung Apps
  - Partner Apps
  - Popular “critical” apps.

Vulnerability Frequency Chart



SAMSUNG

Rodrigo Chiossi

Android Builders Summit 2013

Open Broadcast Receivers

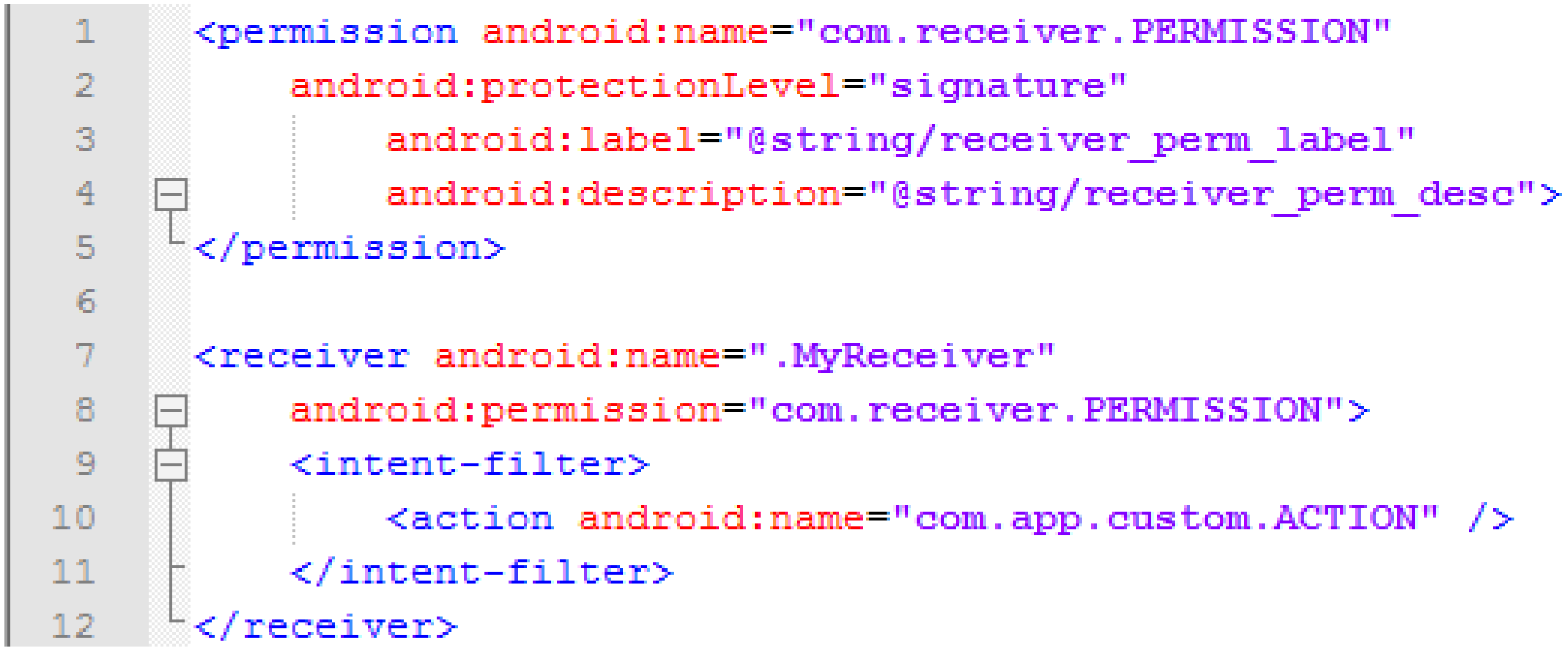
- Occurs when the Broadcast Receiver does not check the source of the intent it received.
- Not usually the intended behavior during development
- Most common use case is to export the broadcast receiver only to a restricted context.
  - E.g. Another app from the same developer.

Default Behavior

- Restricted to the App only
  - Good design choice.
  - Covers the most common use case of Broadcast Receivers.
- When Exported:
  - Default behavior is to be open to everybody.
  - Not the most common use case.

Protection Mechanism

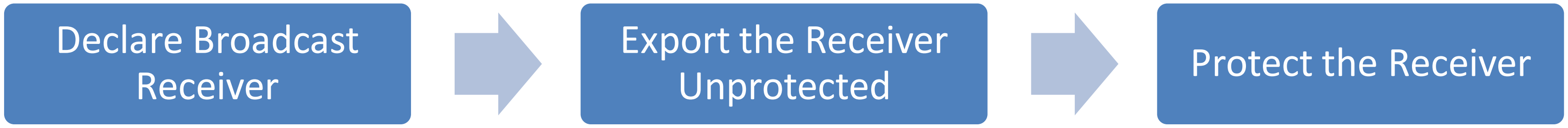
- Protect the Broadcast Receiver with a permission.



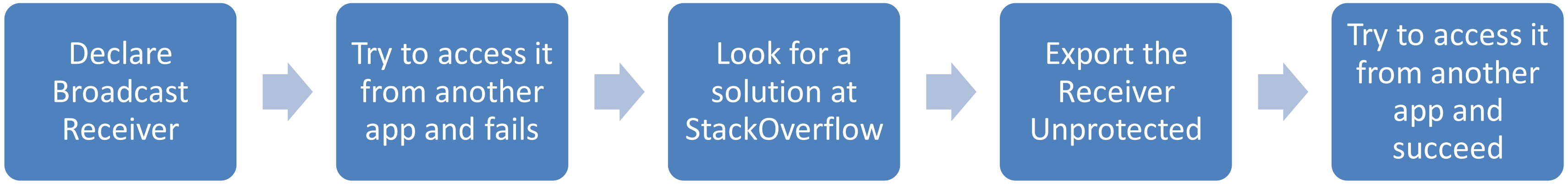
```
<permission android:name="com.receiver.PERMISSION"
    android:protectionLevel="signature"
        android:label="@string/receiver_perm_label"
        android:description="@string/receiver_perm_desc">
</permission>

<receiver android:name=".MyReceiver"
    android:permission="com.receiver.PERMISSION">
    <intent-filter>
        <action android:name="com.app.custom.ACTION" />
    </intent-filter>
</receiver>
```

Implementation Flow

- Proper Implementation Flow

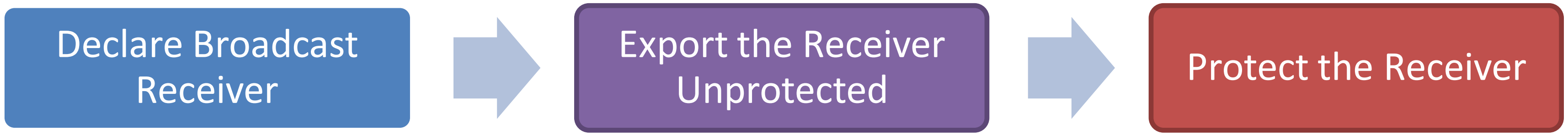


- Developer Implementation Flow

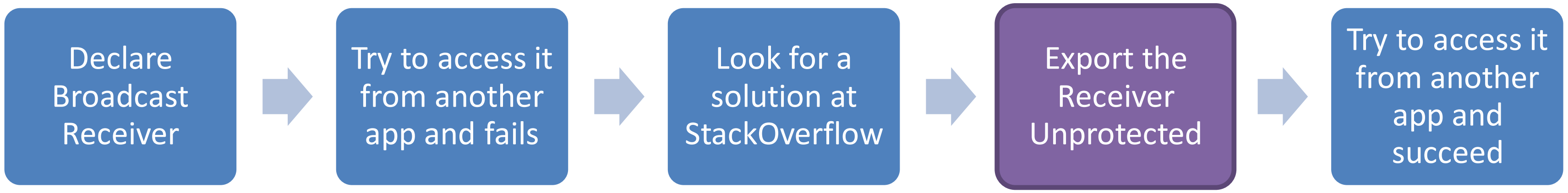

Declare Broadcast Receiver
Try to access it from another app and fails
Look for a solution at StackOverflow
Export the Receiver Unprotected
Try to access it from another app and succeed

Implementation Flow

- Proper Implementation Flow

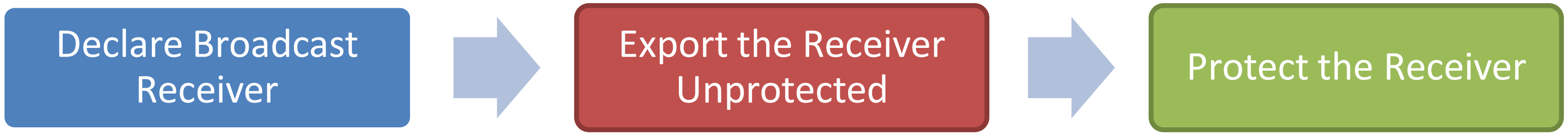


- Developer Implementation Flow

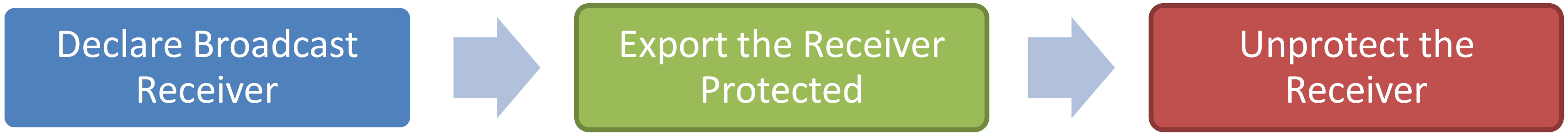

Declare Broadcast Receiver
Try to access it from another app and fails
Look for a solution at StackOverflow
Export the Receiver Unprotected
Try to access it from another app and succeed

Implementation Flow

- Current Implementation Flow

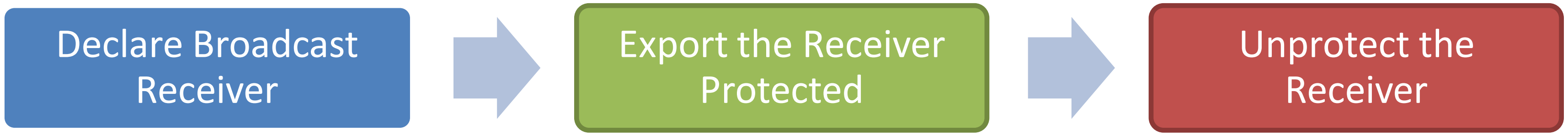


- Healthy Implementation Flow

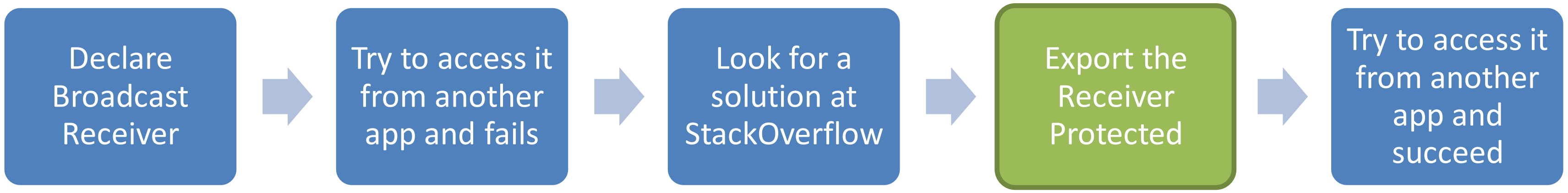


Implementation Flow

- Proper Implementation Flow



- Developer Implementation Flow

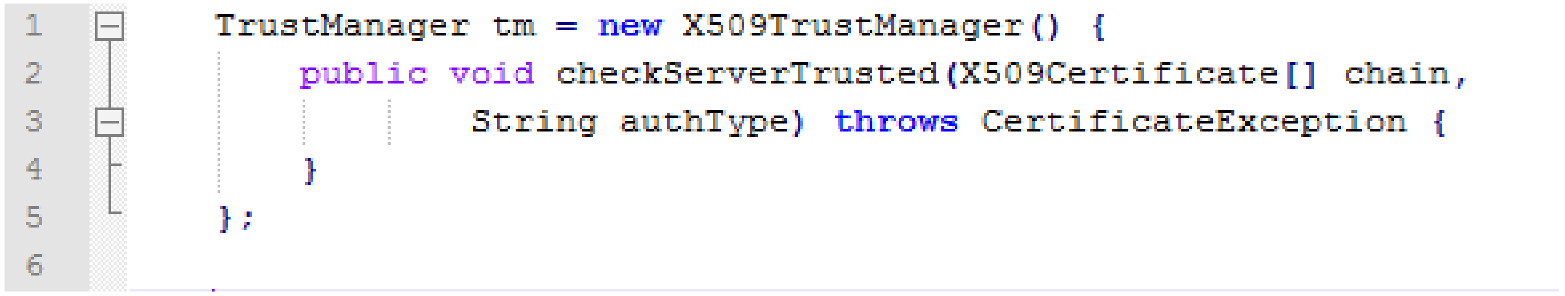

Declare Broadcast Receiver
Try to access it from another app and fails
Look for a solution at StackOverflow
Export the Receiver Protected
Try to access it from another app and succeed

Other applications

- The concept can be applied to other scenarios:
  - Open Services
  - Open Content Providers
- In both scenarios the developer reaches the unprotected state before the protected state.

Improper SSL Handling

- Occurs when the developer validates a self-signed certificate with an empty TrustManager. E. g. :



```java
TrustManager tm = new X509TrustManager() {
    public void checkServerTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
    }
};
```

- Lack of proper documentation and confusing API.
- New version of Android (4.2) already address that issue, but still needs improvement.

The rest of the chart…

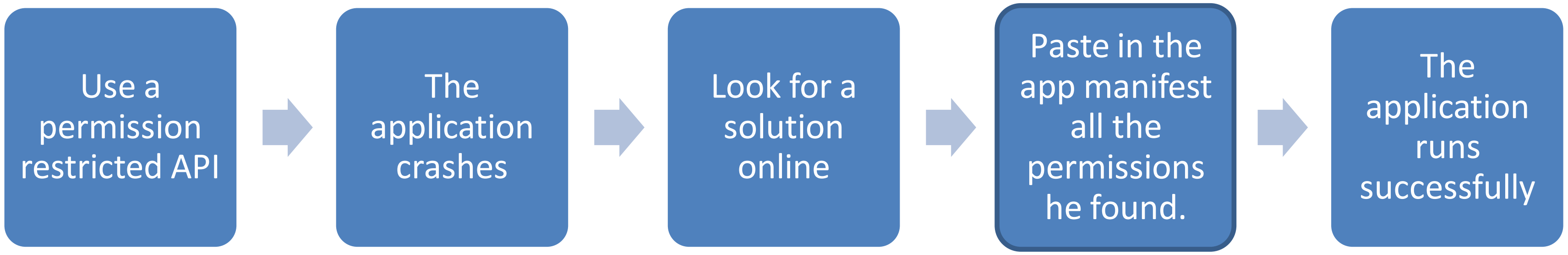
- Other security issues in the chart are developers faults!
- Very bad common habits:
  - Hardcode the crypto key in the application
  - Trust SMS data to perform critical operations

The hidden issue: Excessive Permissions

- Hard to measure with manual assessment.
- Does not introduce a security flaw, but potentialize the risk is one is present.
- The Pwn2Own case:
  - Platform signed application with INSTALL_PACKAGES permission.
  - INSTALL_PACKAGES permission was not required.
  - Enabled an attack to that app to install malicious app in the device.

Permission Declaration Flow

- Developer Implementation Flow


Use a permission restricted API
The application crashes
Look for a solution online
Paste in the app manifest all the permissions he found.
The application runs successfully

Proper Permission Declaration

- A mapping of API-Permission must exist.
- Automate permission declaration for know APIs at compile time.
- Allow for manually add custom permission for unknown APIs.

To Sum Up…

- Not every security issue is a developer's fault.
- It is possible to act directly on the platform to avoid common security problems.
- The developer should always go through the secure state before he is able to reach the insecure state.

Rodrigo Chiossi
r.chiossi@samsung.com
r.chiossi@androidxref.com