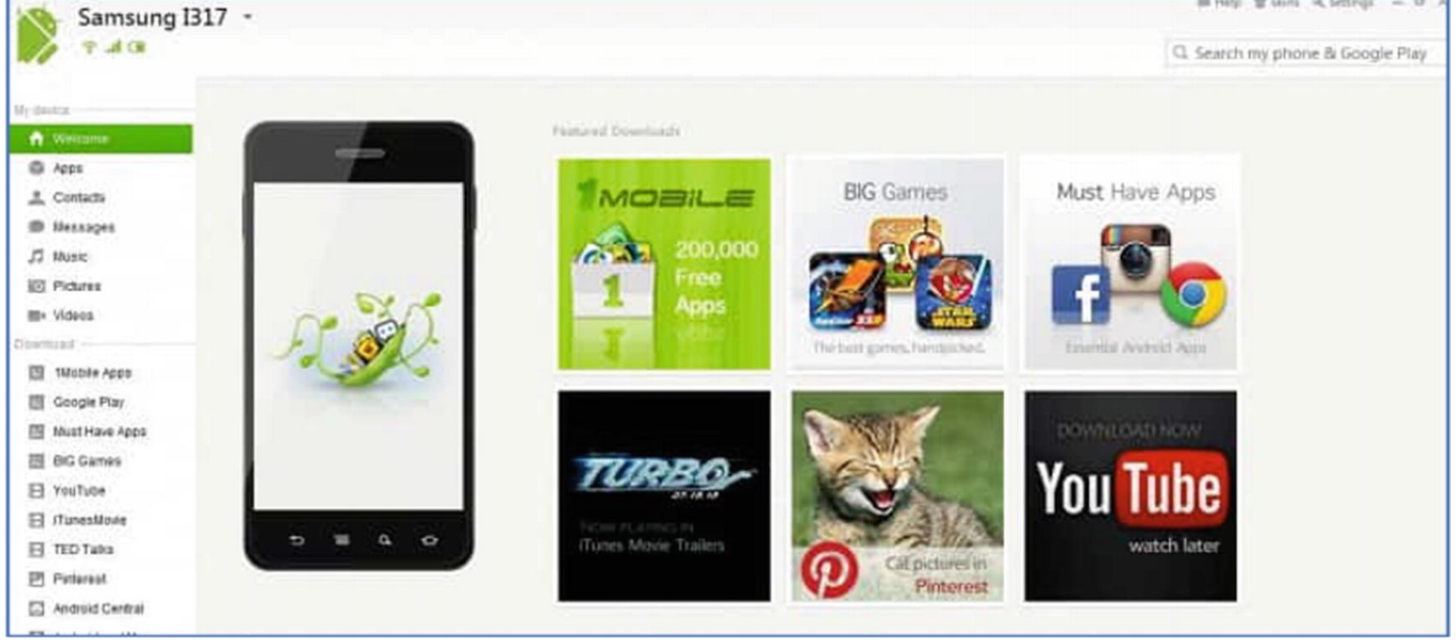
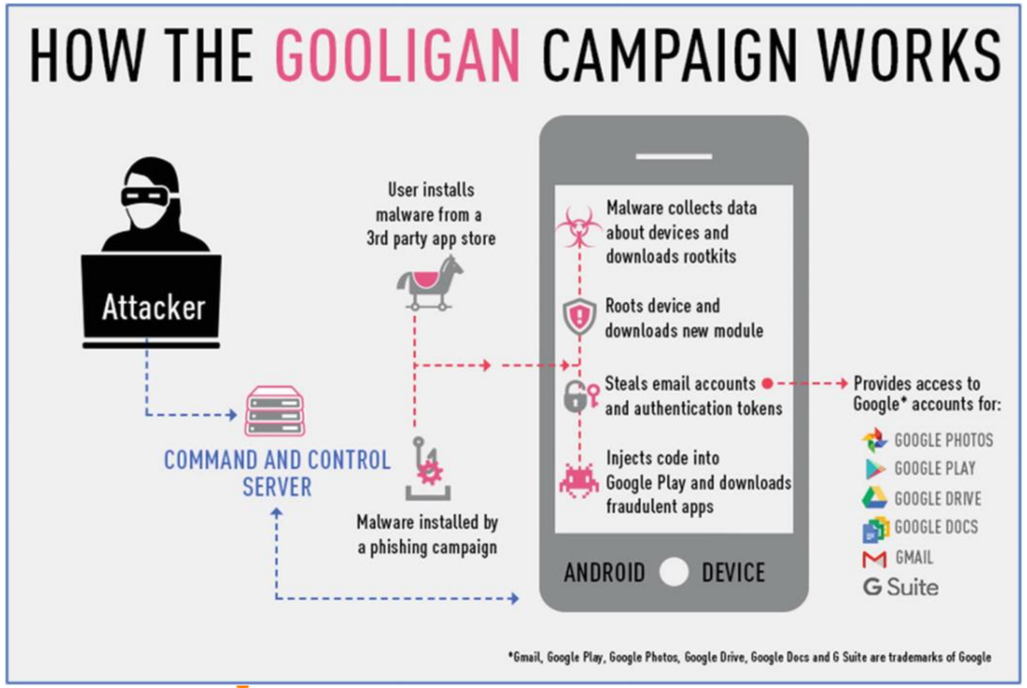
CCT 310 - 3
Application Security

UNIT 4
Android Application Security - I

Current State of Android Application Security



The user interface for SnapPea


HOW THE GOOLIGAN CAMPAIGN WORKS
Attacker
COMMAND AND CONTROL SERVER
User installs malware from a 3rd party app store
Malware installed by a phishing campaign
Malware collects data about devices and downloads rootkits
Roots device and downloads new module
Steals email accounts and authentication tokens
Injects code into Google Play and downloads fraudulent apps
ANDROID DEVICE
Provides access to Google* accounts for:
GOOGLE PHOTOS
GOOGLE PLAY
GOOGLE DRIVE
GOOGLE DOCS
GMAIL
G Suite
*Gmail, Google Play, Google Photos, Google Drive, Google Docs and G Suite are trademarks of Google

Now SnapPea, it was found, would try to root your Android device with 12 different exploits. Check Point has remarked that it was the most amount of Android root attempts that they had witnessed in the wild. Two of the exploits were found to be the vRoot [CVE-2013-6282] exploit and the TowelRoot [CVE-2014-3153] exploit. These two are the only exploits that have CVE IDs in the database. When the rooting of the device was complete, the malware would install several other malicious apps that would further install subsequent ad-laden apps. The malware would then connect to a central server and await further instructions. It seemed like the primary purpose of this malware was to install other adware apps that made all infected devices connect to advertising sites and make it appear like legitimate users were viewing the ads. More views meant more payment for the app owners, and so greater infections yielded greater profits.

Android App Permissions

App permissions help support user privacy by protecting access to the following:
•**Restricted data**, such as system state and users' contact information
•**Restricted actions**, such as connecting to a paired device and recording audio

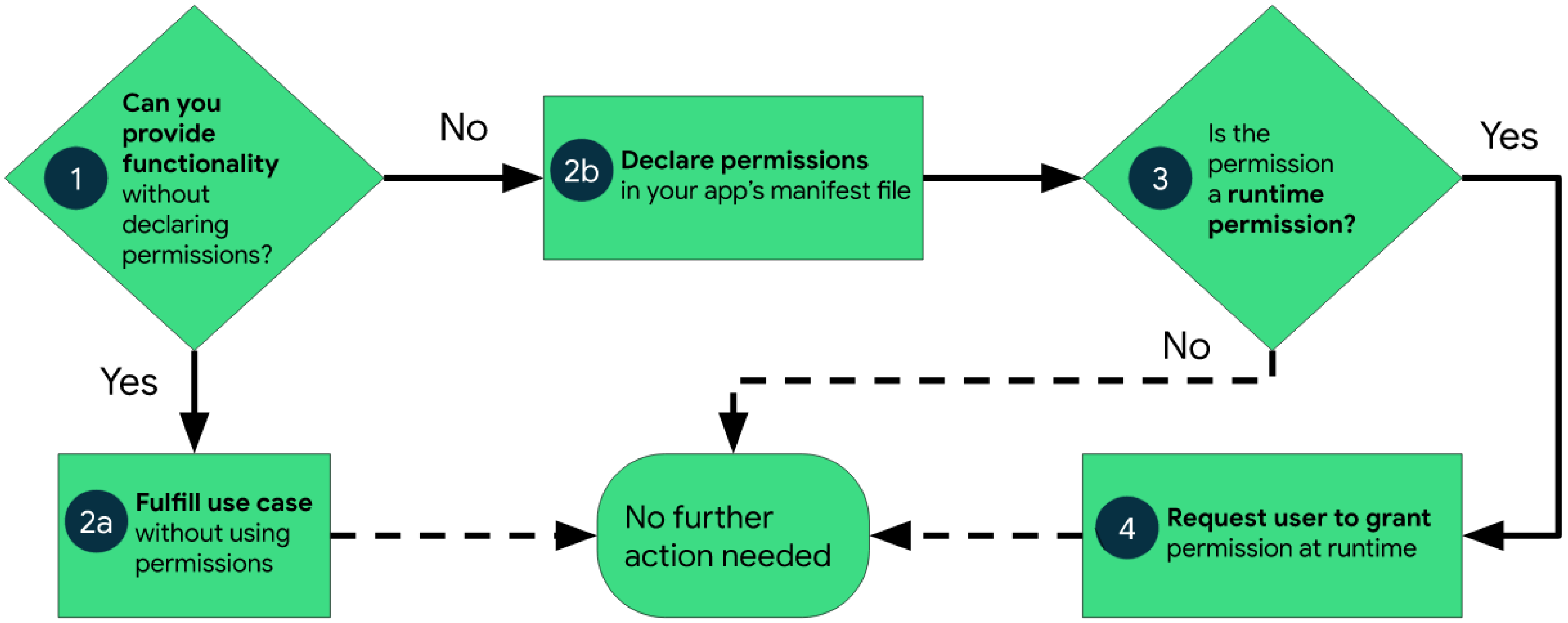
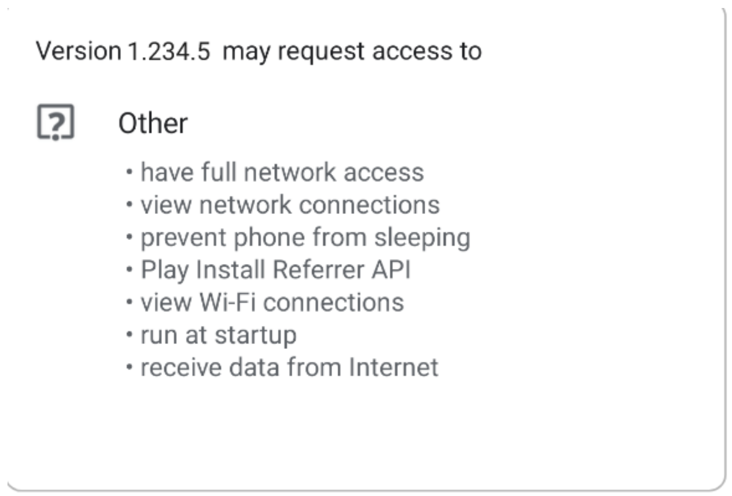

1
Can you provide functionality without declaring permissions?
No
2b
Declare permissions in your app's manifest file
3
Is the permission a **runtime permission?**
Yes
No
Yes
2a
Fulfill use case without using permissions
No further action needed
4
Request user to grant permission at runtime

Figure 1. High-level workflow for using permissions on Android.

Types of permissions


Version 1.234.5 may request access to

Other

- have full network access
- view network connections
- prevent phone from sleeping
- Play Install Referrer API
- view Wi-Fi connections
- run at startup
- receive data from Internet

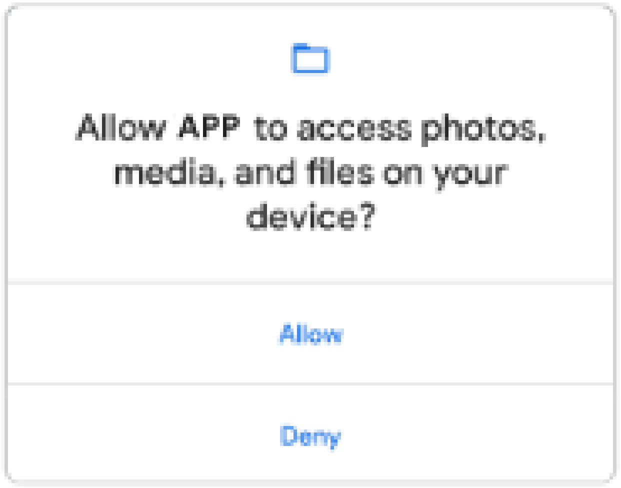
Install-time permissions give your app limited access to restricted data or let your app perform restricted actions that minimally affect the system or other apps. When you declare install-time permissions in your app, an app store presents an install-time permission notice to the user when they view an app's details page, as shown in figure. The system automatically grants your app the permissions when the user installs your app.

Normal permissions

These permissions allow access to data and actions that extend beyond your app's sandbox but present very little risk to the user's privacy and the operation of other apps.
The system assigns the normal protection level to normal permissions.

Signature permissions

The system grants a signature permission to an app only when the app is signed by the same certificate as the app or the OS that defines the permission.
Applications that implement privileged services, such as autofill or VPN services, also make use of signature permissions. These apps require service-binding signature permissions so that only the system can bind to the services.


Allow APP to access photos, media, and files on your device?

Runtime permissions, also known as dangerous permissions, give your app additional access to restricted data or let your app perform restricted actions that more substantially affect the system and other apps. Therefore, you need to request runtime permissions in your app before you can access the restricted data or perform restricted actions. Don't assume that these permissions have been previously granted—check them and, if needed, request them before each access.

When your app requests a runtime permission, the system presents a runtime permission prompt, as shown in figure.

Many runtime permissions access *private user data*, a special type of restricted data that includes potentially sensitive information. Examples of private user data include location and contact information.

The microphone and camera provide access to particularly sensitive information. Therefore, the system helps you explain why your app accesses this information.

The system assigns the dangerous protection level to runtime permissions.

Special permissions

Special permissions correspond to particular app operations. Only the platform and OEMs can define special permissions. Additionally, the platform and OEMs usually define special permissions when they want to protect access to particularly powerful actions, such as drawing over other apps.

The **Special app access** page in system settings contains a set of user-toggleable operations. Many of these operations are implemented as special permissions.

Learn more about how to request special permissions.

The system assigns the appop protection level to special permissions.

Permission groups

Permissions can belong to permission groups. Permission groups consist of a set of logically related permissions. For example, permissions to send and receive SMS messages might belong to the same group, as they both relate to the application's interaction with SMS.

Permission groups help the system minimize the number of system dialogs that are presented to the user when an app requests closely related permissions. When a user is presented with a prompt to grant permissions for an application, permissions belonging to the same group are presented in the same interface. However, permissions can change groups without notice, so don't assume that a particular permission is grouped with any other permission.

Best practices to be followed for Security & Privacy for building Android Applications :-

App permissions build on system security features and help Android support the following goals related to user privacy:

- **Control:** The user has control over the data that they share with apps.
- **Transparency:** The user understands what data an app uses and why the app accesses this data.
- **Data minimization:** An app accesses and uses only the data that's required for a specific task or action that the user invokes.

Request a minimal number of permissions
When the user requests a particular action in your app, your app should request only the permissions that it needs to complete that action. Depending on how you are using the permissions, there might be an alternative way to fulfill your app's use case without relying on access to sensitive information.

Associate runtime permissions with specific actions
Request permissions as late into the flow of your app's use cases as possible. For example, if your app lets users send audio messages to others, wait until the user has navigated to the messaging screen and has pressed the **Send audio message** button. After the user presses the button, your app can then request access to the microphone.

Consider your app's dependencies
When you include a library, you also inherit its permission requirements. Be aware of the permissions that each dependency requires and what those permissions are used for.

Be transparent
When you make a permissions request, be clear about what you're accessing, why, and what functionalities are affected if permissions are denied, so users can make informed decisions.

Make system accesses explicit
When you access sensitive data or hardware, such as the camera or microphone, provide a continuous indication in your app if the system doesn't already provide these indicators. This reminder helps users understand exactly when your app accesses restricted data or performs restricted actions.

Basic Android Testing

Benefits of testing

Testing is an integral part of the app development process. By running tests against your app consistently, you can verify your app's correctness, functional behavior, and usability before you release it publicly.

You can *manually* test your app by navigating through it. You might use different devices and emulators, change the system language, and try to generate every user error or traverse every user flow.

However, manual testing scales poorly, and it can be easy to overlook regressions in your app's behavior. *Automated testing* involves using tools that perform tests for you, which is faster, more repeatable, and generally gives you more actionable feedback about your app earlier in the development process.

•**Functional testing**: does my app do what it's supposed to?
•**Performance testing**: does it do it quickly and efficiently?
•**Accessibility testing**: does it work well with accessibility services?
•**Compatibility testing**: does it work well on every device and API level?

•**Unit tests** or **small tests** only verify a very small portion of the app, such as a method or class.
•**End-to-end** tests or **big tests** verify larger parts of the app at the same time, such as a whole screen or user flow.
•**Medium tests** are in between and check the **integration** between two or more units.

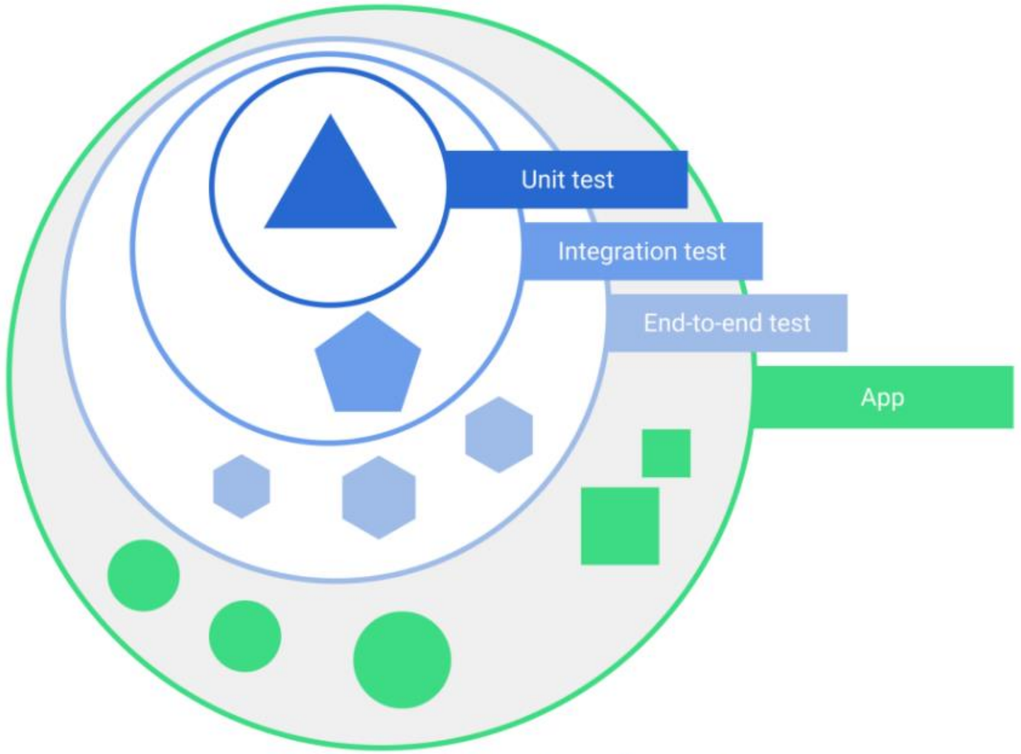
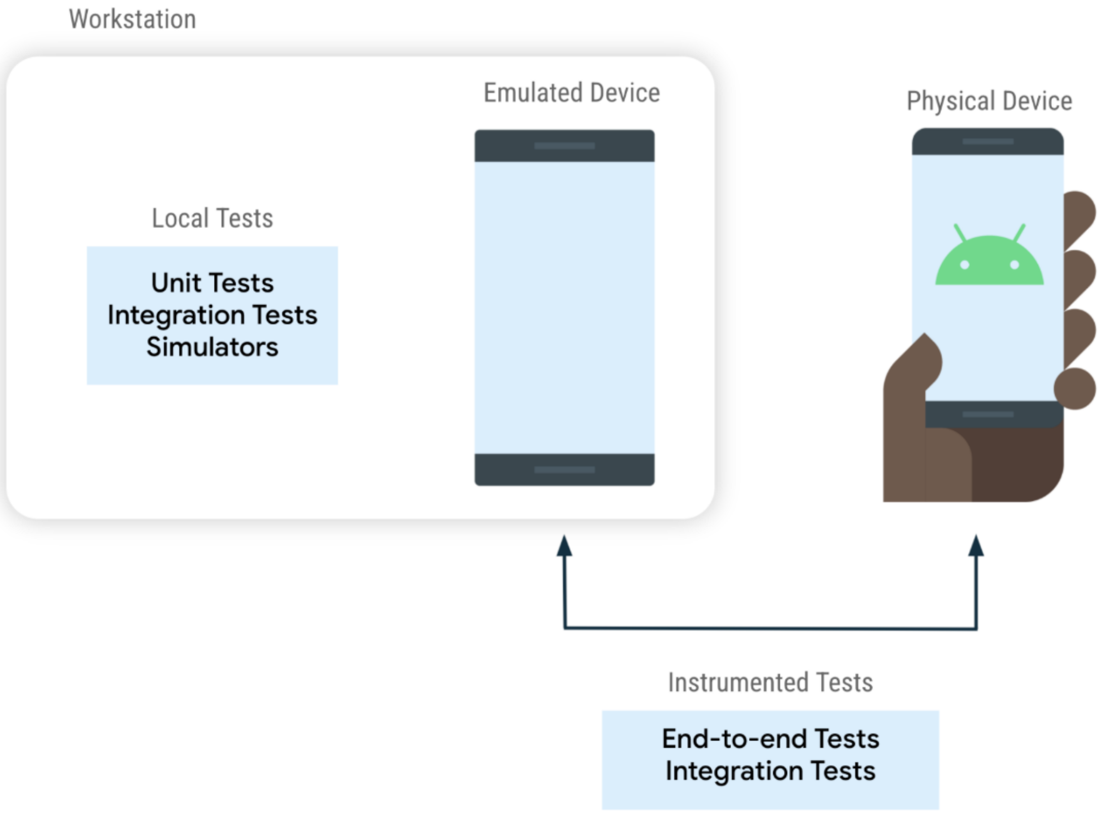


Figure 1: Test scopes in a typical application.

- **Instrumented tests** run on an Android device, either physical or emulated. The app is built and installed alongside a *test app* that injects commands and reads the state. Instrumented tests are usually UI tests, launching an app and then interacting with it.
- **Local tests** execute on your development machine or a server, so they're also called *host-side tests*. They're usually small and fast, isolating the subject under test from the rest of the app.



OWASP Mobile Top 10

https://owasp.org/www-project-mobile-top-10/

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality

Hacking Android Apps

https://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html

https://ibotpeaches.github.io/Apktool/

https://github.com/topics/android-exploit

The Three Biggest Threats to Android Devices

Threat One: Data in Transit

Mobile devices, including those running Android as an operating system, are susceptible to man-in-the-middle attacks and various exploits that hack into unsecured communications over public Wi-Fi networks and other wireless communication systems. By hijacking a user's signal, attackers can impersonate legitimate web services, steal data, or intercept calls and text messages.

Threat Two: Untrustworthy App Stores

Untrustworthy app stores can cause headaches due to lack of security protocols. Ensure that your app store of choice for Android applications takes adequate security precautions and has a strong security review program in place. Sideloading, in which you install apps without an app store, is also a process to manage carefully due to a lack of foundational security measures.

Threat Three: SMS Trojans

Malicious apps can sometimes include SMS trojans, which come in the form of compromised applications. This type of app accesses a mobile device's calling or text message capabilities, allowing them to do things like send text messages with malicious links to everyone in a user's address book. These links can then be used by attackers to distribute computer worms and other malicious messages to fee-based services, incurring fees on behalf of the user and profiting scammers.

Three Ways to Protect Your Android Devices

Use TLS Encryption

OWASP shows that insufficient encryption is a big problem for many types of applications. By using Transport Layer Security (TLS), you can encrypt internet traffic of all types for securely generating and exchanging session keys. This protects data against most man-in-the-middle and network spying attacks.

Test Third-Party App Security

The best way to avoid malicious apps is to only use apps from the official Google Play store. Google Play uses significantly better security checks than third-party sites, some of which may contain hundreds of thousands of malicious apps. If you absolutely need to download an app from a third-party store, check its permissions before installing, and be on the lookout for apps which that for your identity or the ability to send messages to your contacts when they don't need to.

Use Caution When Using SMS Payments

Set your Android phone to limit the ability of apps to automatically spend your money. Apps that ask for payment via SMS are a red flag and should be avoided if at all possible.