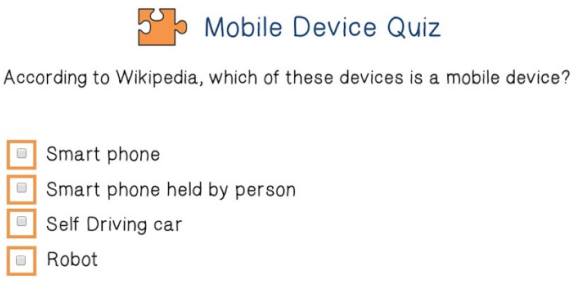
This lesson incorporated materials from:

John Mitchell at Stanford University & Yanick Fratantonio at UC Santa Barbara.

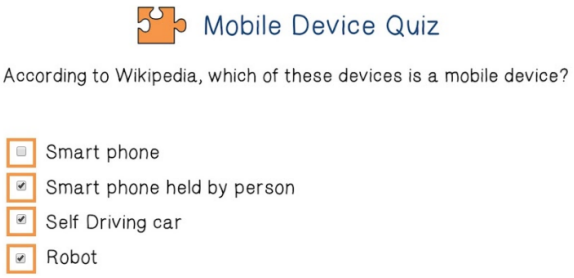
Jekyll on iOS: When Benign Apps Become Evil.
Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, and Wenke Lee.
In Proceedings of The 22nd USENIX Security Symposium. Washington DC. August 2013.

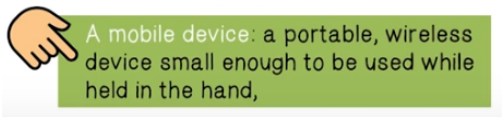
You can find the summary of the paper at the end of the document.



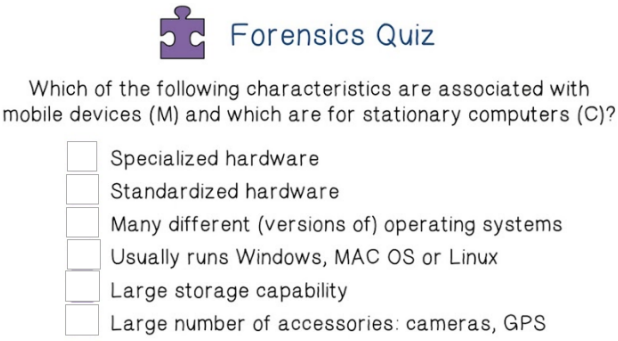
We are going to spend this lesson discussing mobile malware or malware on mobile devices. So let us take a moment to make sure that we all agree as to what we mean when we say mobile device. According to Wikipedia, which of these devices is a mobile device?

Wikipedia says that a mobile device must be mobile. Therefore, a smartphone by itself is not mobile. However, a smarthone held by a person can be mobile because it is a non mobile device with a mobile host. These are examples of true mobile devices because either myself can be mobile.

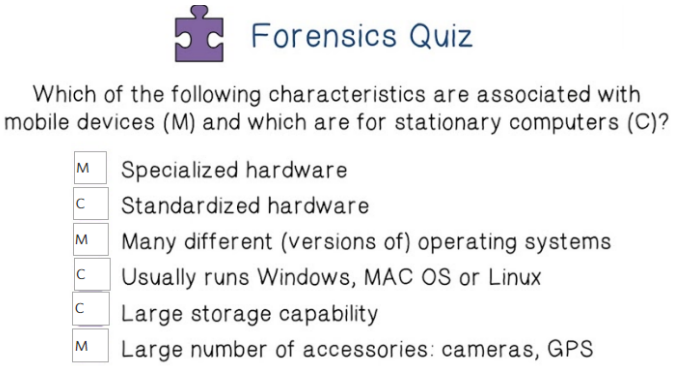




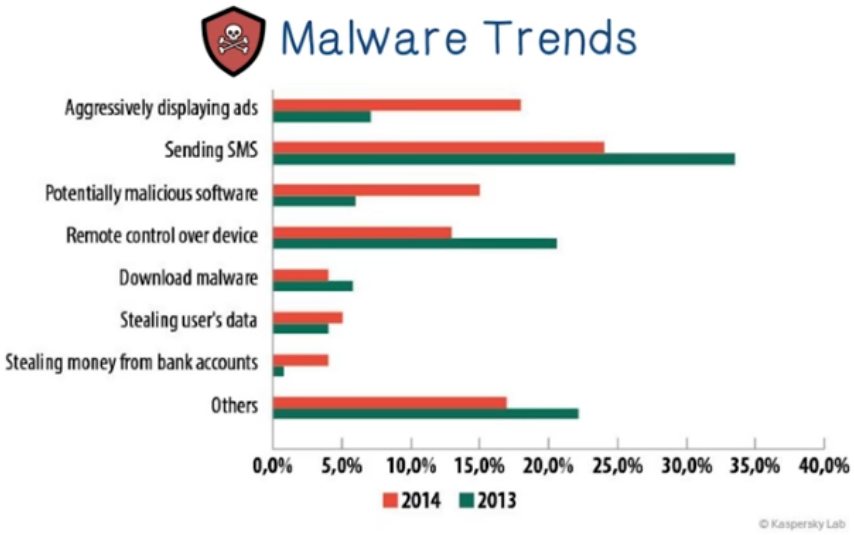
We are not going to use the strict definition of mobile device. Instead, we are going to use this common definition. That is a smartphone is a mobile device.



Before we discuss mobile malware, we should understand the difference between mobile devices and the traditional stationary computers. Which of the following characteristics are associated with mobile devices versus stationary computers?

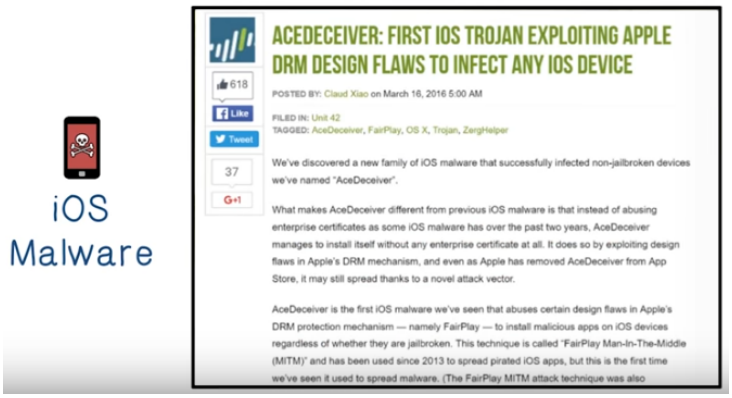


Mobile devices tend to use specialized hardware. Whereas stationary computers tend to use standardized hardware. Mobile devices tend to use many different versions of operating systems. For example, there are many different versions of Android. Whereas stationary computers tend to run Windows, MAC OS or Linux. They also have large storage capacity. On the other hand, mobile devices tend to have a large number of accessories such as cameras and GPS.



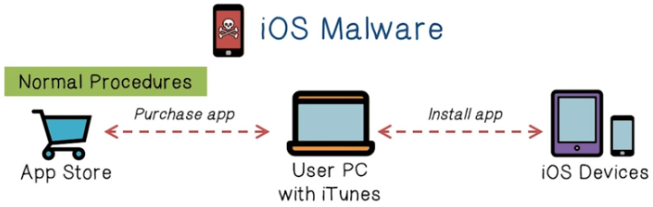
Since mobile devices are increasingly used for critical functions in our daily lives, and they have become powerful computers with good connectivity, they have become security targets. This part shows the major categories of mobile malware.

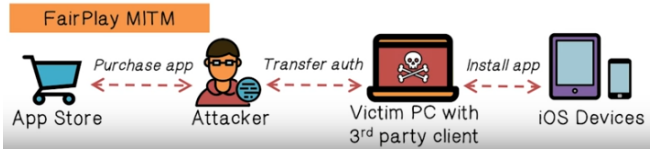
Some apps may appear to be providing useful functions, but they secretly still use confidential information. These apps are considered malware. Here is an example of such malicious apps and it was taken off the app store.



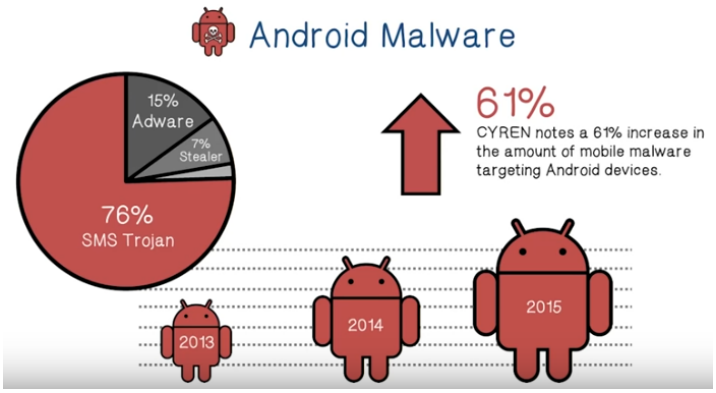
But malicious apps may still get on victims devices even after it is taken off the app store. One technique is to exploit the design floor in Apple's DRM scheme.

Apple allows users to purchase and download iOS apps from their app store through the iTunes client running on their computers. They can then use their computers to install the apps onto their iOS devices. The iOS devices will request an authorization code for each app installed to prove the app was actually purchased.



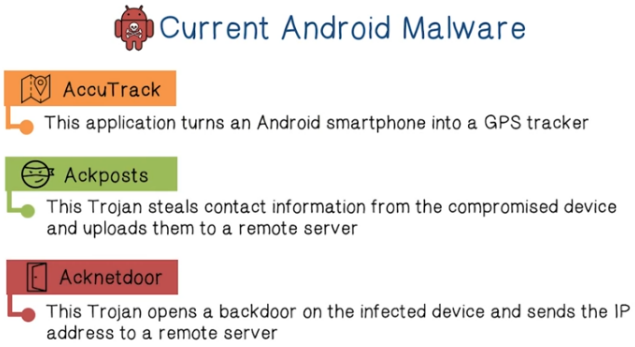


In a FairPlay man-in-the-middle attack, attackers purchase an app from the app store, then intercept and save the authorization code. They then develop a program that simulates the iTunes software, and installs the software on a victim's computer. And this fake iTunes software can check the victim's iOS devices to believe the app was purchased by the victim. Therefor the user can install apps they never actually paid for, and the attacker can install malicious apps without the user's knowledge. This attack continues to work even after the malicious app is removed from the app store.



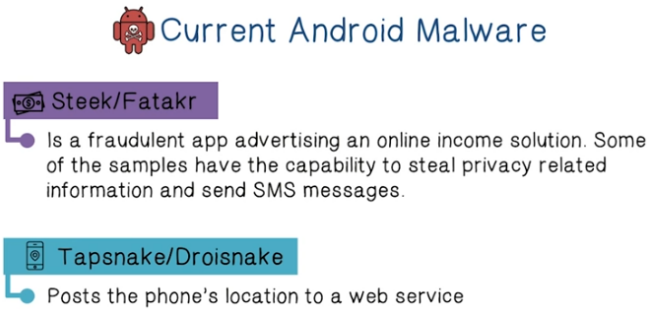
There is a large increase of the number of Android malware and the majority is still SMS Trojans. You can read more about current Android malware at the link in the instructor's notes.

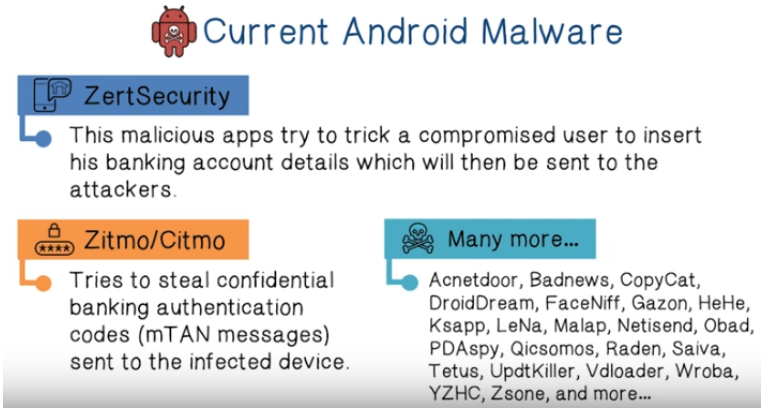
Instructor Notes: Android Malware



We discuss a few here. AccuTrack turns an Android smartphone into a GPS tracker. Ackposts steals contact information. Acknetdoor opens a backdoor.

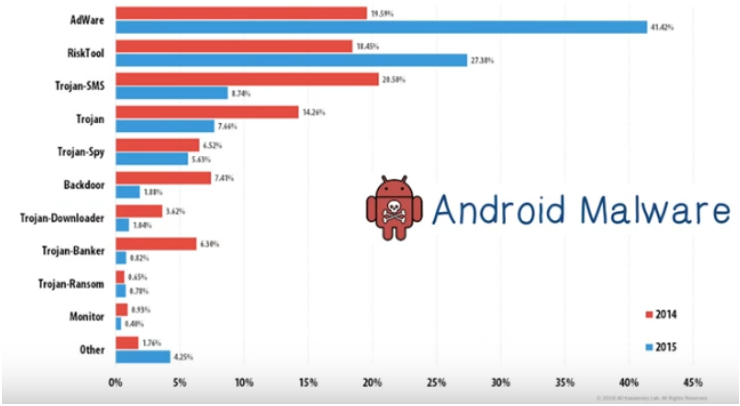
Steek does fraudulent advertising and also steals private information. Tapsnake posts the phone's current location to a web service. ZertSecurity steals the user's bank account information.





Similarly, Zitmo also steals the user's bank account information. And there are many other more. Again, you can read more about the current Android malware at the link in the instructor's notes.

This plot summarizes the major categories of Android malware and their trends. There are quite a few free Android antivirus apps, here are a few examples.



"Even security companies know the risk is low — that's why apps are packaged with other selling points." - AndroidCentral

"Symantec sees an important role to play in helping to protect data and mobile devices from being exposed to risk..."

The security companies own statements provide some insights of the state of anti-malware on mobile phones. First of all, the risk is relatively low. In general, mobile devices are still less powerful than desktops and laptops and there are plenty of those that can be targeted by hackers. Furthermore, the app review process and the sandbox space execution model also means that the mobile devices are in general more secure. Nevertheless, there is still the need to protect mobile devices.

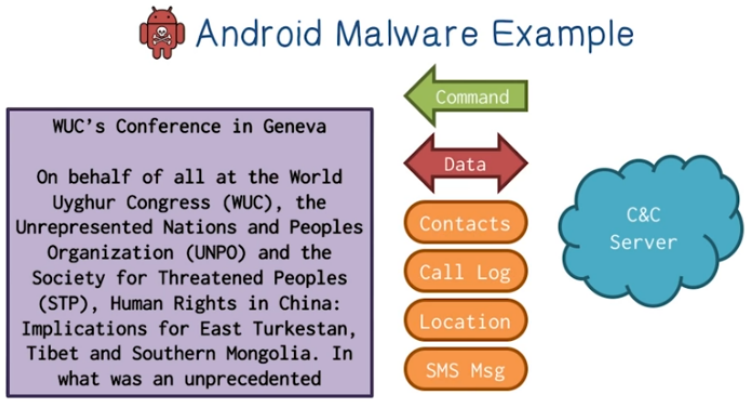
In particular, in addition to protection against malware, a bigger problem is the loss of devices and the loss or the theft of data. And therefore secure companies also try to protect and manage mobile data.

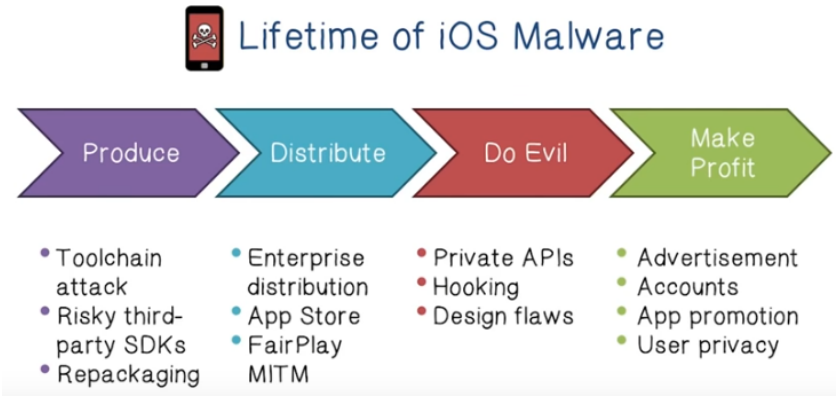
"While Symantec sees its purpose in the mobile landscape as providing security against malware, fraud and scams; we also protect devices against loss and theft — loss of the device itself, as well as the information on it. In addition, Symantec helps businesses protect and manage their data being stored or transmitted through the mobile devices of their employees."

Mobile malware are becoming more sophisticated and are showing the same advanced features of malware and laptops and desktops. Here is an example of Android malware that uses social engineering to spread. It targets a group of activists. On March 24, 2013, the email account of a high-profile Tibetan activist was hacked and it was used to send spear phishing emails to people on his contact lists. This is what the spear phishing email looked like. The recipient of this email is tricked that he should install this APK file on his Android device which is the malware.

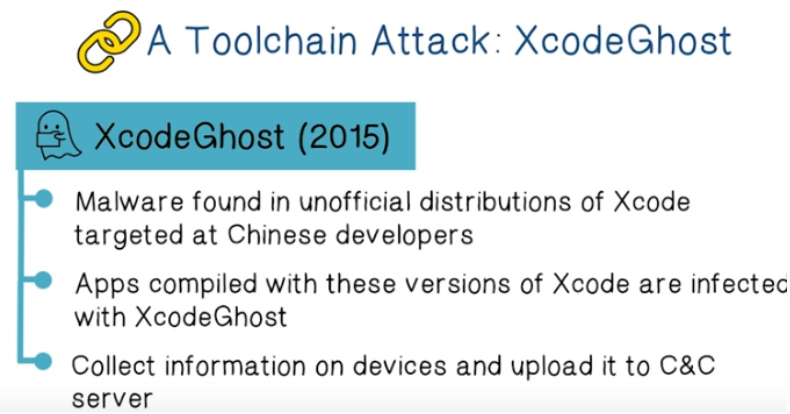
After installation, an application named Conference appears on the desktop. After the installation, if the user launches this Conference app, he will be seeing the information about the upcoming event.

While the victim reads this fake message, the malware secretly reports the infection to a command control server. After that, it begins to harvest information stored on a device. The stolen data includes contacts, call logs, SMS messages, your location and phone data, which includes phone number, OS version, phone model, etc. Note that the stolen data will not be uploaded to the command-and-control center automatically. The malware waits for incoming SMS messages from the command-and-control center. And based on the message, the malware knows what data to upload.



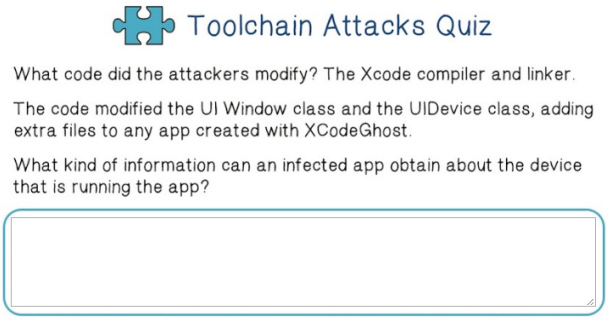


Now let us review iOS Malware, in particular the lifetime or stages of an iOS malware and how the functions of each stage can be realized. These stages include produce, distribute, do evil and make profit. And there are multiple approaches to go about at each stage.

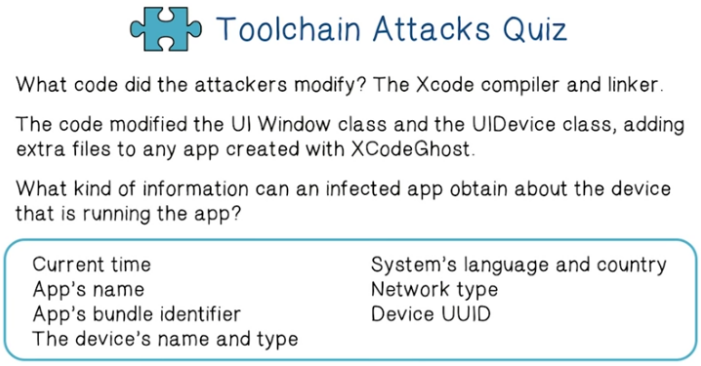


Toolchain attack is one approach to produce malware. Here is a real example, an official distribution of Xcode was compromised with malware. If a developer used this infected Xcode library to develop an app, the app will be infected. The infected app will then collect information on devices and upload the data to a command-and-control (C&C) server. This attack is very potent because any app that is compiled using this Xcode library now becomes a malicious app. This XcodeGhost was able to infect many apps, including 39 apps published in the official iOS App Store.

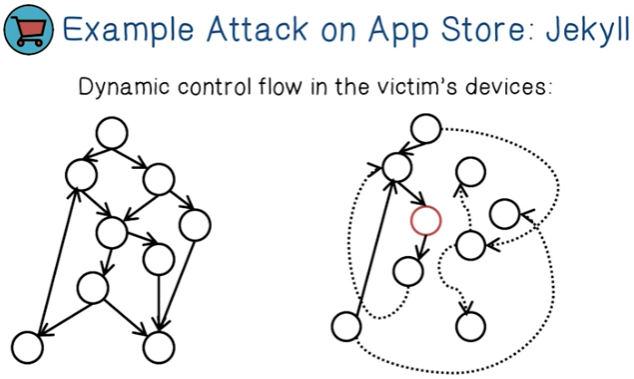
You can read more about the XCodeGhost attack with the link in the instructor's note. Then you can answer the question, what kind of information can an infected app obtain from the device?



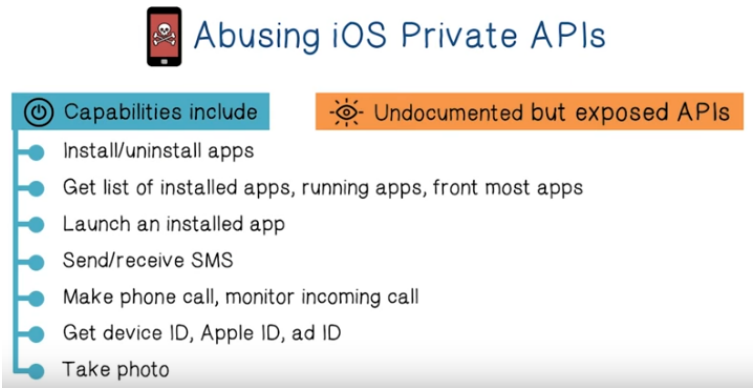
There is quite a bit of information that can be gathered by the infected app. Information can then be used to craft further attacks, or it can be used to steal passwords and usernames.



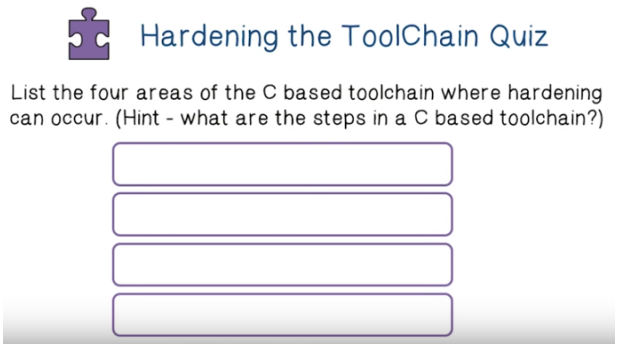
Attacking the App Store review process is one approach to distribute malware. Here is an example. We created the Jekyll app in 2013, and planted vulnerability in this app. This vulnerability can be exploited at runtime with a particular input. Once the vulnerability is exploited the Jekyll app can activate new additional execution paths through return-oriented programming. And then the app can send SMS, email, tweet, and so on. On the other hand, the App Store review process cannot find these malicious paths, because they cannot review without a correct input and runtime.



To illustrate, the App Store review process finds that the control flows of the app to be safe, that is, all the execution paths are acceptable. On the other hand, a runtime vulnerability planted in the code is exploited because of the specific input, and as a result, the app uses new control flows that were not observable in the app view process.

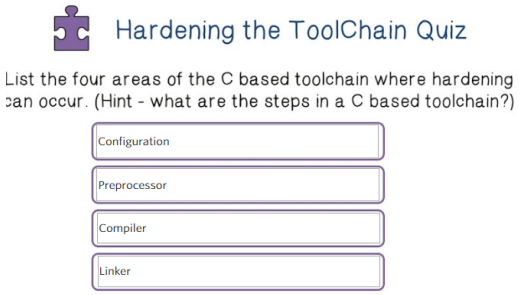


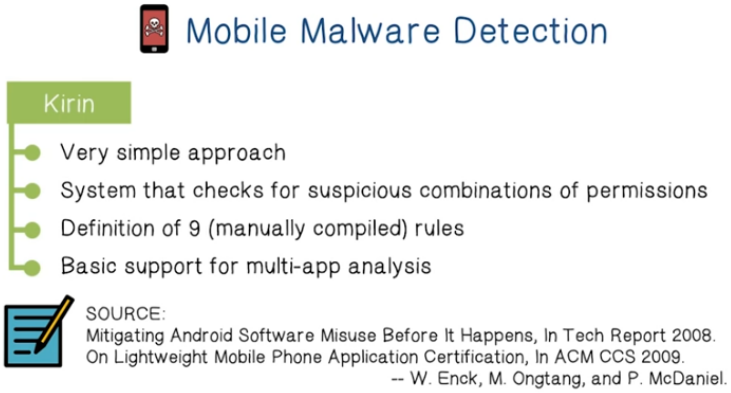
These new control flows allow the Jekyll app to do evil things and make profit. For example, it can perform a number of activities. These can be achieved by calling private APIs that are not directly accessible to legitimate apps. But a Jekyll app knows the memory layout and hence the addresses of these APIs and can directly jump to them.



We now know that toolchain attacks have occurred and will continue. So, the question is, can you hold on the toolchain? In this quiz, list the four areas of the C based toolchain where hardening can occur.

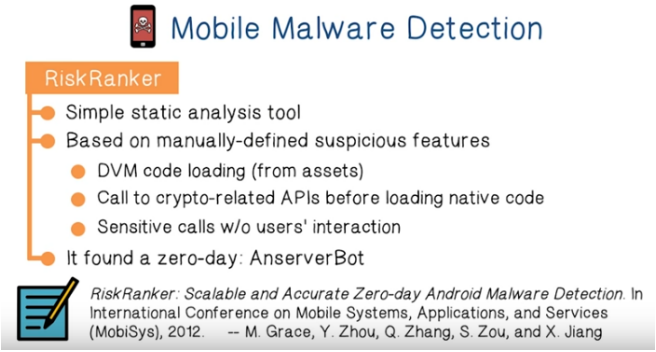
Essentially, all the main steps of the toolchain can be hardened.

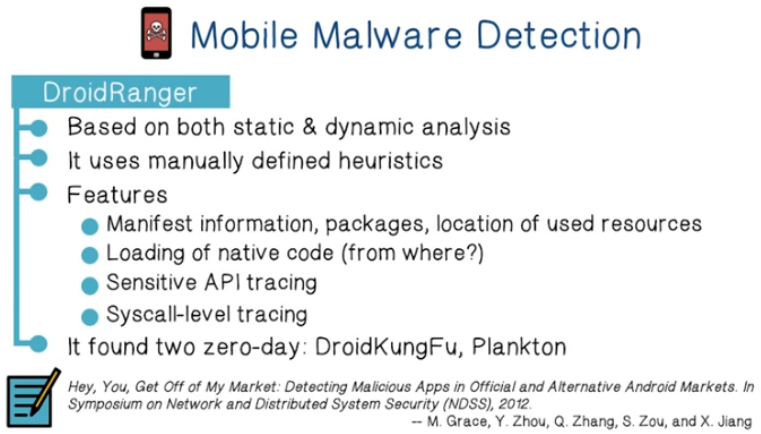




Kirin is a very simple system that looks for suspicious combination of permissions.

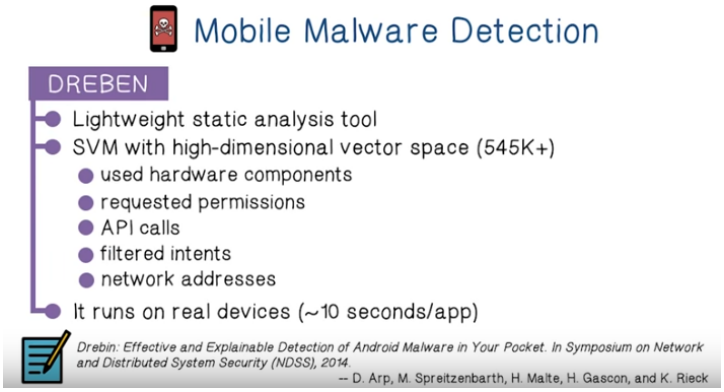
RiskRanker use heuristics such as cryptos that relates to unpacking code.

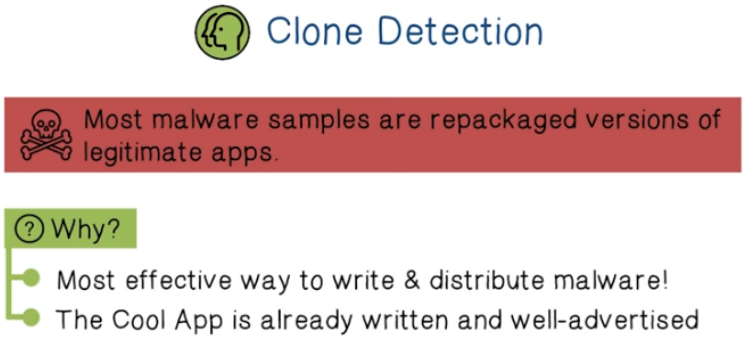




Similarly, DroidRanger use heuristics such as loading native code from suspicious websites.

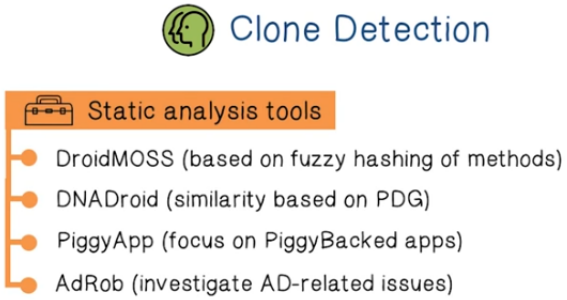
DREBEN uses a machine learning algorithm called SVM, or Support Vector Machine. And the data attributes used for modeling include permissions, API calls, and so on.

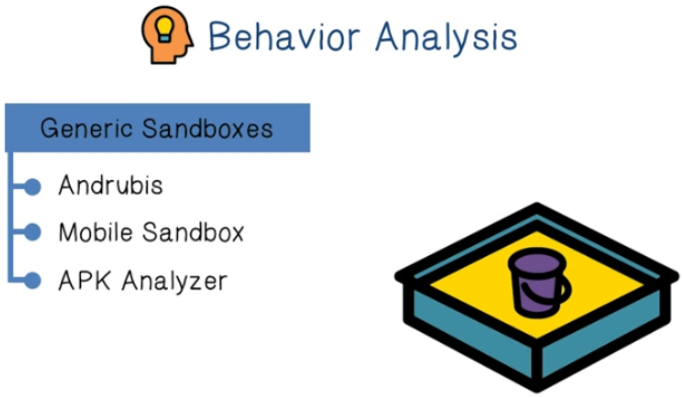




Many malicious apps are actually repackaged version of legitimate apps. This is actually the most effective way to distribute malware, because a popular or cool app already has a large number of users.

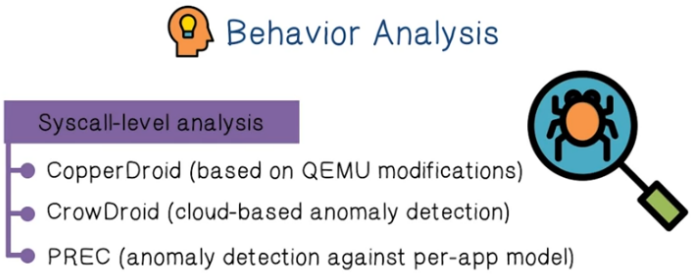
There are research systems on clone detection. Here are a few examples. For example, DroidMOSS uses fuzzy hashing of Java methods to match and detect clone code. DNADroid performs similarity analysis on PDGs. PDGs are program dependency graphs between methods.

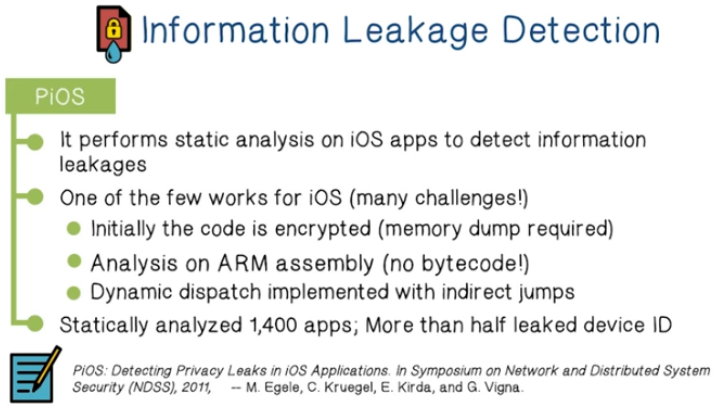




There are a few sandboxes for mobile malware analysis. And these sandboxes enable dynamic analysis. Here are a few example sandboxes.

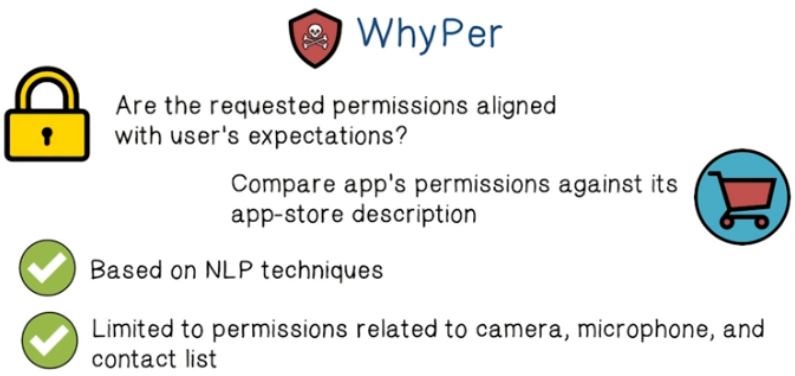
Many dynamic analysis and detection tools use system call information. Here are a few examples. For example, PREC stands for particle root explore containment. It can dynamically identify system calls from high risk components, for example, third-party native libraries, and execute those system calls within isolated threads. Therefore, PREC can detect and stop root exploits with high accuracy, while imposing very low interference to benign applications.





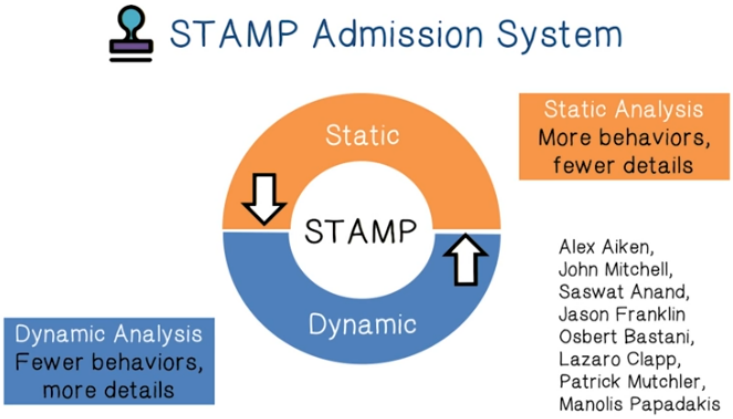
Information leakage is a big concern, and apps that leak sensitive information can often be considered malware even though many think that they are in a gray area. There are several approaches to detect leakage, for example, PiOS performs static analysis to detect information leakage.

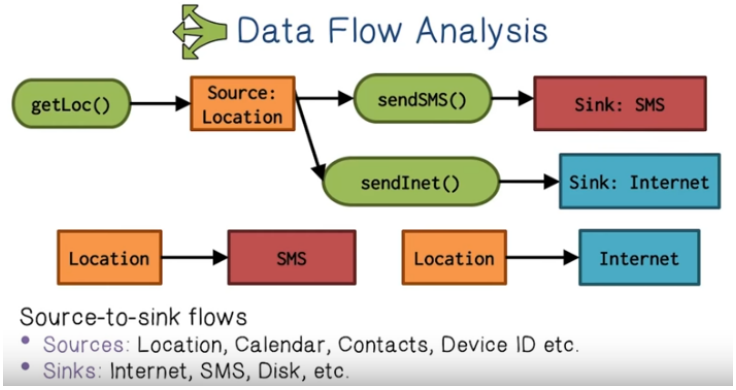
TaintDroid uses taint tracking to perform information flow analysis. That is, it analyzes how data from a source, such as address book, flows to the sync, such as the internet.



Another approach is to check if an app does what it promises to do, for example, WhyPer compares the app's permissions against its description. And the analysis is based on natural language processing techniques.

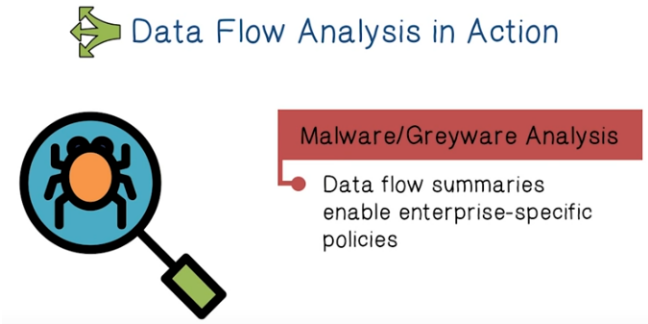
Now, let us discuss a research system designed to analyze mobile apps and decide if the mobile apps meet security and privacy requirements. The system is intended to be used in an App Store to decide if an app should be admitted. The system uses both static and dynamic analysis approaches because they have pros and cons of their own.

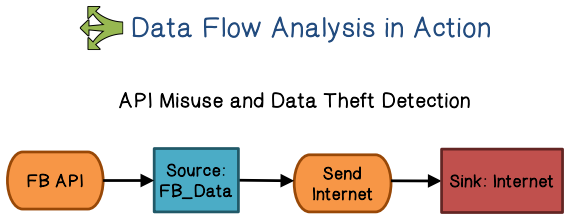




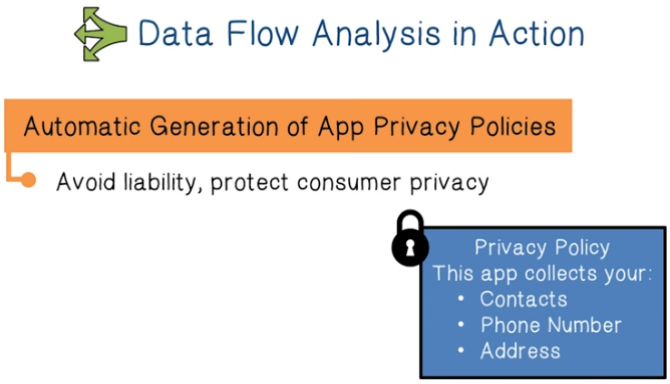
One of the most important analysis is data flow analysis. Here is an example of data flow, the source is location data and the sink is SMS or website on the internet.

Data flow analysis can be useful for malware or greyware analysis to find out what information is being stolen. And based on the discoveries, we can improve enterprise specific policies.



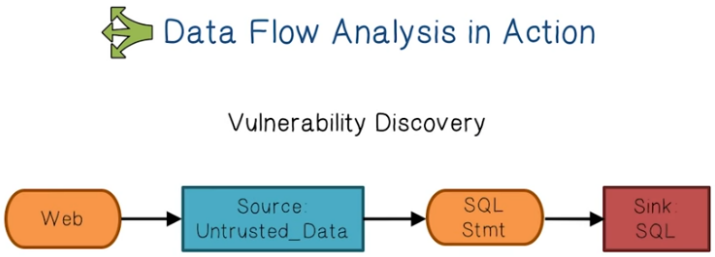


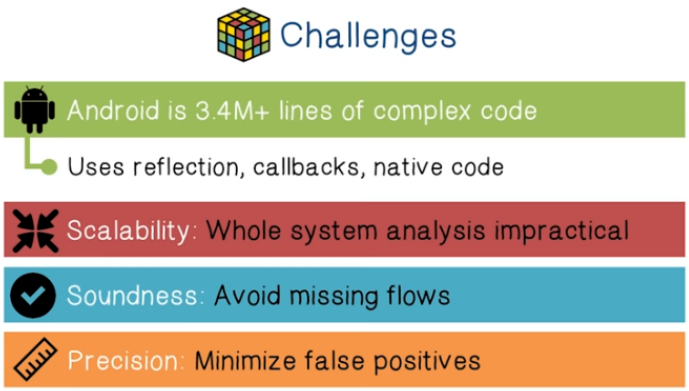
Data flow analysis can be used to check the external app to make sure that there is no API abuse or data theft.



Data flows gathered from an app can be used to inform users about potential privacy implications.

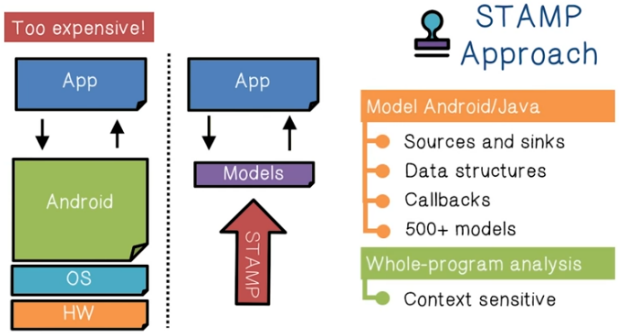
Data flow analysis can also be used to discover abilities in applications. For example, accepting data from untrusted sources.

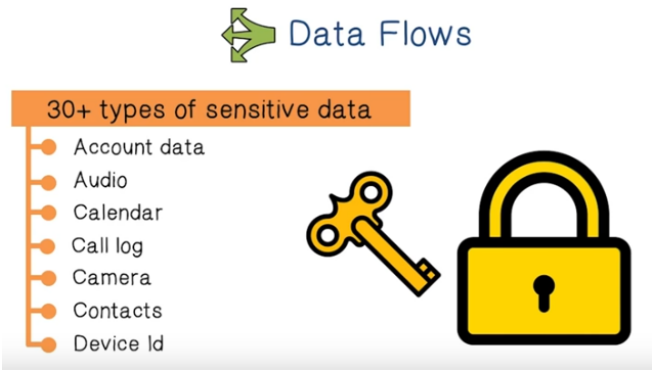




However, analyzing data flows is a very challenging task. For example, Android has more than three million lines of very complex code. Performing data analysis on the whole system would take a long time, and it is not practical. And of course to be useful, data flow analysis has to be accurate.

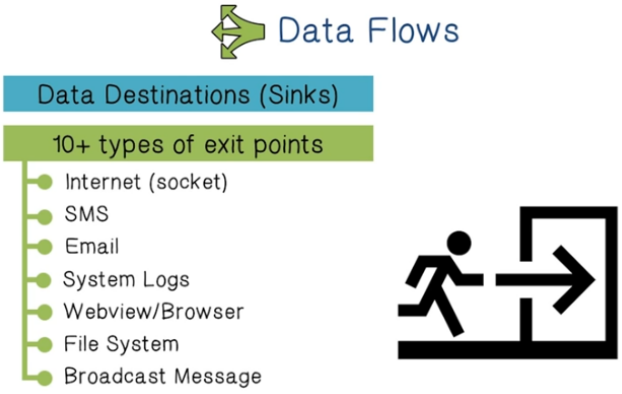
As we have just discussed, analyzing an app in a context of full Android is very expensive because there is too much code involved. The STAMP approach is to abstract the Android stack into models. And these models include the following information. We are going to focus on data flows.

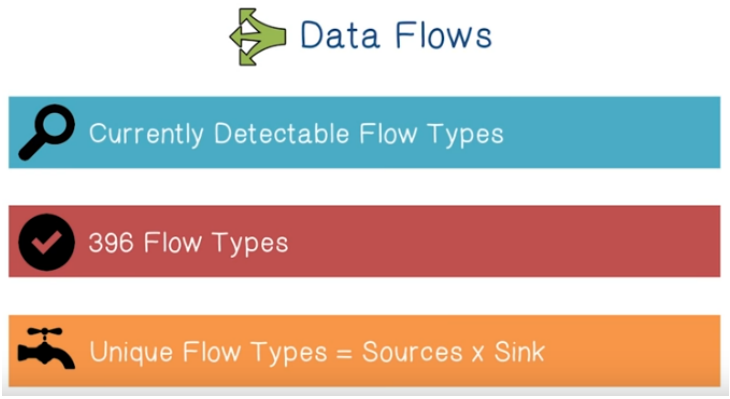




There are more than 30 types of sources. Here are some examples.

There are more than ten types of sinks, and here are some examples.

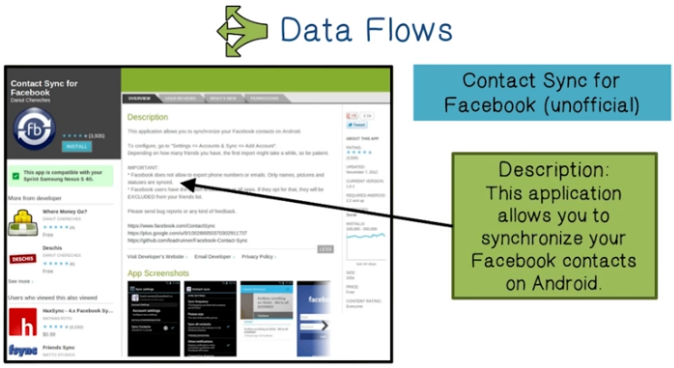


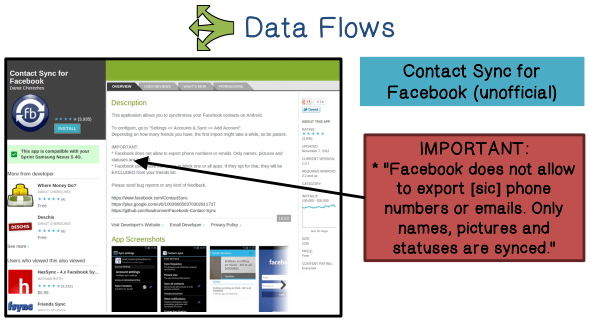


Each pair of source and sink is a flow type, and there are close to 400 flow types.

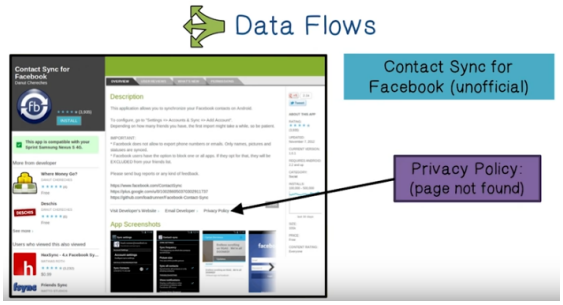
Here is an example of data flow analysis on the Facebook app.

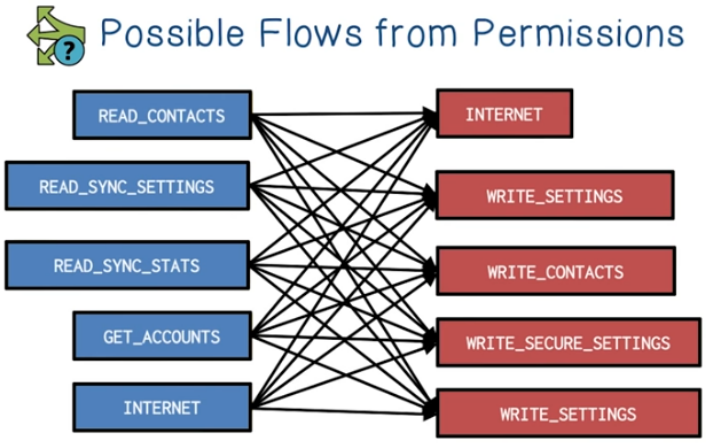
The description of the app says that it allows the user to synchronize contacts. And it says that Facebook does not allow the export of phone numbers or emails.





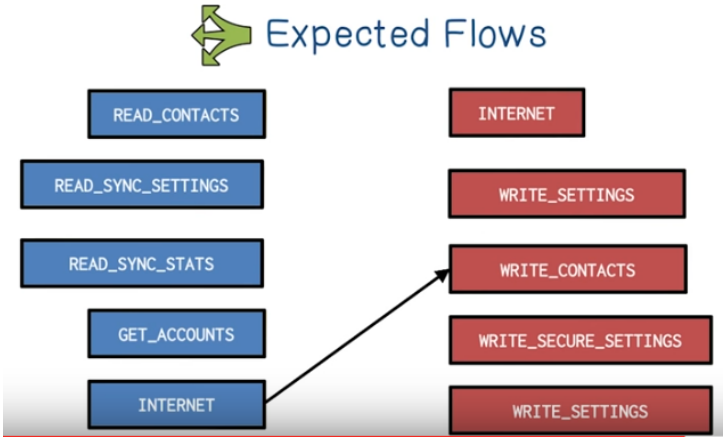
And the users can block one or all apps but there is no privacy policy.

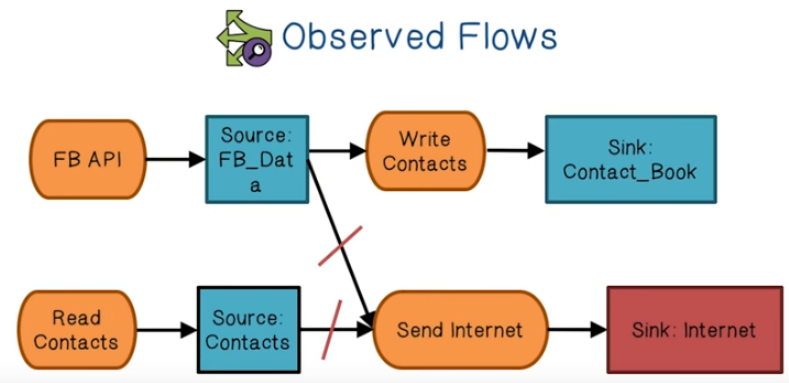




Here are the possible flows. On the left, we have the sources. On the right, we have the sinks. That is, potentially all resources can go to all the sinks.

From the Facebook description, we expect to see the state of flow for sinking contacts, and here is an example.





However, the data flows observed from the Facebook app include additional flows that lead to leakage.

Jekyll on iOS: When Benign Apps Become Evil

This paper introduces a new class of iOS apps, termed Jekyll Apps, which have the capability to maneuver around Apple's mandatory app review and code signing mechanisms which ensure that only approved apps can run on iOS devices. The researchers showed that by designing an app to be remotely exploitable and subsequently introducing malicious control flows by rearranging signed code after the app has been installed, it can slip through the vetting process and perform many malicious tasks, such as stealthily posting tweets, taking photos, stealing device identity information, sending email and SMS, attacking other apps, and even exploiting kernel vulnerabilities. It is important to note, however, that the developed attack does not assume any specifics about how Apple reviews apps, but targets theoretical difficulties faced by any known methods to analyze programs, thus serving to highlight the shortcomings of the pre-release review approach and incentivize more runtime monitoring mechanisms to protect iOS users in the future.

The basic approach to making a Jekyll App is to plant remotely exploitable vulnerabilities in a normal app, decompose the malicious logic into small code gadgets and hide them under the cover of the legitimate functionalities. After the app passes the App Review and lands on the end user device, the attacker can remotely exploit the planted vulnerabilities and assemble the malicious logic at runtime by chaining the code gadgets together.

The researcher's give a high-level overview of the Security features of iOS devices, specifically the features of Data Execution Prevention (DEP), Address Space Layout Randomization (ASLR), Privilege Separation and Sandboxing, Code Signing, App Store and App Review. To defeat the ASLR, DEP, and code signing mechanisms in iOS, and at the same time be hardly detectable, the researchers designed an information leakage vulnerability through which the app deliberately leaks its partial runtime memory layout information to the remote attacker. Thus, the attacker can infer the locations of the pre-deployed gadgets, making ASLR useless. Next, they planted a buffer overflow vulnerability in the app through which the attacker can smash the stack layout and hijack the app's control flow. To avoid the vulnerabilities from being detected in the review process, the communication between the app and the server was encrypted, and all the vulnerabilities had special trigger conditions. Finally, the hidden gadgets were discretely distributed in the app and mingled with the normal functionalities, without explicit control flow or and data flow connections via means of creating a number of infeasible branches across the entire code space and hiding gadgets under these infeasible branches.

All these steps combined to facilitate a host of malicious behaviors carried out on the victim devices via the Jekyll App. These behaviors ranged from posting tweets, sending Emails and SMS, sneakily turning on the camera and recording videos, commandeering the Bluetooth Device, and dialing arbitrary numbers. Furthermore, it also enabled the researchers to be able to exploit Kernel Vulnerabilities. Finally, despite iOS Sandboxing, and given the available of IPC protocols between sandboxed apps using

URL scheme handlers, the researchers were able to attack other apps that have vulnerabilities when handling malformed URL requests.

Finally, the researchers close by pointing out the infeasibility of depending on the App Store Screening Process to catch this class of malicious apps. They do point out that it is more dependable to combat these apps (though not easily) by improving the existing security mechanisms or introducing more advanced runtime monitoring mechanisms can limit Jekyll apps' capability to perform malicious operations.