Examples for Research Project Milestone 1

Prof. William Enck

February 7, 2021

1 Example for Empirical Study

1.1 PoliCheck

Note: This example project proposal corresponds to Andow et al., "Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck", In Proceedings of the USENIX Security Symposium, 2020.

Title: Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck

Research Question: Privacy is a long-standing open research challenge for mobile applications. Literature has proposed various program analysis tools for Android and iOS apps, often citing private-information disclosure as motivations. However, just because an application sends privacy-sensitive information to the Internet does not mean there is a privacy violation. A recent idea proposed in research is to compare the *flows* of privacy sensitive information observed for an application to the *language* in the application's privacy policy. While there have been some studies that have attempted to measure this *flow-to-policy consistency*, their measurement was limited in that is not sensitive to the *entity* receiving the information. For example, they assume that if a privacy policy states the app will share sensitive data to *any* party (first or third party), then it is okay to send to *any* party (first or third-party). This assumption may falsely claim that an app is consistent with its policy if, for example, the policy states that only the first-party (app developer) will receive a type of data, but the app also sends the data to an advertiser. This leads to the following research question:

Does entity-sensitive *flow-to-policy consistency identify more inconsistent mobile applications than entity-insensitive analysis, and if so, what is the impact of the previous over-estimation?*

Proposed Methodology: We will formally define an *entity-sensitive* flow-to-policy consistency model for mobile applications. We will then build upon two prior tools: (1) PolicyLint, which extracts data sharing statements from privacy policies, and (2) AppCensus, which dynamically analyzes mobile apps to identify when privacy sensitive information is sent to the network. We will apply these tools to a large set of free mobile applications and their privacy policies, which we will download from the Google Play Store. Using this data, we will answer the above research question.

Expected Findings: We expect to find that without entity-sensitive analysis, prior flow-to-policy consistency measurements have significantly underestimated the problem, and that many applications send information to third-parties (e.g., analytics servers) that are not mentioned in the app's privacy policy.

2 Example for Building a Solution

Note: This example proposal corresponds to Nadkarni et al., “Practical DIFC Enforcement on Android”, In proceedings of the USENIX Security Symposium, 2016.

Title: Practical DIFC Enforcement on Android

Problem: Application-based modern operating systems, such as Android, thrive on their rich application ecosystems. Applications integrate with each other to perform complex user tasks, providing a seamless user experience. To work together, applications share user data with one another. Such sharing exposes the user’s private and enterprise information to the risk of exfiltration from the device. Unfortunately, Android’s permissions are only enforced at the first point of access. Once data is into the memory of an untrusted application, it can be exported.

The classic way to address this problem is using Information Flow Control (IFC). Unfortunately, incorporating IFC into commodity operating systems has been historically challenging, because applications often deal with many different types of information during an execution, leading to poor information tracking precision and subsequently a phenomenon commonly known as label explosion or label creep. Several prior works have identified that Android’s runtime model, which is based on collections of event-driven components (e.g., activity components), may be particularly amenable to IFC. However, attempts of incorporating IFC into Android have been limited, frequently breaking backwards compatibility by improperly killing processes or blocking until an application voluntarily exits, which can lead to deadlocks.

Solution Idea: We propose a novel solution to IFC in Android called *lazy polyinstantiation*. The key idea is that our approach manages instances of each application, their components, and their storage for each secrecy context from which an application is called.

Solution Approach: We will create a formal model for incorporating lazy polyinstantiation into Android and then modify the Android Open Source Project (AOSP) with logic that encodes our model. We will then formally evaluate the security of our model, as well as empirically evaluate the compatibility of our modified Android using a collection of real applications downloaded from the Google Play Store.

Expected Findings: We expect that our lazy polyinstantiation model will be secure. We expect that our model and implementation will be compatible with existing applications. For example, it will not require improperly killing processes or blocking. Finally, we expect the performance overhead of our implementation to be minimal.