Section 2: Week 3: Mobile Security Literature Review

Nate Bachmeier

TIM-7010: Computer Networking and Mobile Computing

July 14, 2019

North Central University

# Mobile Security Literature Review

## Extending E-Business Applications Using Mobile Technology (2006)

As the iPhone began to take off around 2006 businesses started to envision the future scenarios that mobility would bring. Consider the scenario where a customer wants to purchase a new toaster. While at the store they can reach into their pocket and instantly harness reviews and recommendations on this purchase. If they cannot find the item in stock, then the sales representative reaches into their pocket and in a few clicks requests additional inventory to fill the order.

These connected scenarios will transform every aspect of the supply chain, as each participant is ‘Always on and Always Connected’ (AO/AC) to the corporate data services.

## Software Engineering for Mobility: Reflecting on the Past, Peering into the Future (2014)

Shortly after 2006 businesses started to realize the vision of AO/AC scenarios. The delays were caused by (1) the screens being too small, (2) network connectivity too slow, and (3) inconsistent support for design languages such as Hyper-Text Markup Language (HTML).

This forced businesses to create multiple interfaces for their websites as a partial solution to supporting both platforms. To further complicate matters the lack of consistency between device vendors resulted in scenarios where *multiple* mobile interfaces needed to be written and maintained. These challenges impacted the broad adoption of mobility computing.

However, the devices exploded in popularity and forced the carriers to provide faster connectivity. Their omnipresence made it impossible for retailers and online services to ignore support for these platforms.

## Android Permissions Demystified (2011)

The market leader within mobile operating systems is Google’s Android, an open source solution (OSS) that is based on the Linux kernel. Android supports multiple applications performing separate workloads on the same physical device.

These workloads, called Android Packages, are primarily installed through the Google Play Store. The Store is an open ecosystem that allows developers to publish their applications with minimal scrutiny. This has led to a rich environment full of novelties, games, and unfortunately malware.

Integrity of the device is maintained through a policy-based solution that is based on SE Linux. By default, all rights are denied and need to be approved by the user at installation time. The expectation is that involving the end user in the authorization process should discourage apps from requesting dangerous rights.

However, in practice this is difficult as the non-technical users are being asked to make technical decisions. Even technical audiences, such as Android developers, can lack the rich understandings required to choose the correct permissions causing additional attack surface.

## Research on Android Access Control based on Isolation Mechanism (2016)

Android permissions are difficult to correctly configure as they are ‘coarse-grained authorization and permission models.’ For instance, an app that manages the configuration of *Bose Bluetooth Head Phones* needs to have rights to all Bluetooth devices.

There have been efforts over the years to allow the end users to selectively enable subsets of permissions on an application. Though these efforts are rarely successful as development teams do not support partial trust scenarios. To partially mitigate this scenario, Android 6 introduced the notion of runtime prompting if a dangerous system API is called.

## Figment: Fine-grained Permission Management for Mobile Apps (2019)

It can be argued that correctly handling fine grained permissions is technically possible today. The patterns have been well documented and implementing them is simply a matter of business priority. The implementation requires tedious boiler plate code that needs to be duplicated across the code base.

The authors of Figment believe that Aspect Oriented Programming is best suited for this scenario. Their library exposes attributed for annotating the security requirements of different classes and methods in the code base. During compilation these attributes are used to inject cross cutting concerns as needed.

The injected code will request the activation of any missing permission, just in time. When users decline the request, Figment determines if its possible to continue with partial trust. This information is obtained during the compilation phase based on call graph analysis.

## Investigating User Perception and Comprehension of Android Permission Models (2018)

A critical design aspect of the Android permission system is the assumption that users can make well informed security decisions. Usability studies attempt to quantify this assertion by surveying hundreds of users. These users often understand that a request for ‘SMS Permissions’ is referring text messaging but cannot differentiate tell whether the permission will read or write them.

Another observation is that granted permissions are hard to recall after the fact. It has been proposed that permissions are granted at runtime, not at installation time. To provide more context the dialog could provide local examples of other apps that have those same rights.

The researchers failed to address that users could associate the *dangerous* permission with the *safe* applications. For example, if the dialog said that Google Chrome also uses the permission, and they trust Google to keep them safe – then it must be safe to allow the action.

## Role-based Privilege Isolation: A Novel Authorization Model for Smart Devices (2011)

If the user cannot reliably make security critical decisions, then the system needs to not require them. Role Based Privilege Isolation creates multiple distinct personas on the device, and then prohibits cross role sharing. Perhaps Alice has a smart phone that is used for work, school, and leisure. She could create three roles and then assume different roles for different contextual functions.

There are many technical strengths to this approach, however it is difficult for users to recall which context an application is under. Perhaps Alice launched Chrome for work, then Bob called to discuss dinner reservations. If she does correctly transition into the leisure role, then there is cross role contamination.

## An Efficient Implementation of Next Generation Access Control for Mobile Health (2018)

An argument can be made that Alice was doomed to failure because she is attempting to operate with three coarse roles. A more fine-grained solution exists with Policy Machines (PM). A Policy Machine is represented as a Directed Acyclic Graphs (DAG); with each entity (e.g., user, file, resource) within the system is expressed as nodes and rights as edges.

Using traditional graph algorithms, the policy state can be inspected without requiring extensive resources. That includes auditing scenarios such as determining after the fact what apps are permitted to use which resources.

## The design of graph-based privacy protection mechanisms for mobile systems (2019)

Another advantage of centralizing control policy within a graph, is that additional complex relationships can be mined. Researchers propose the notion of ‘elevation of privacy’ paths which combine multiple harmless rights to devise an aggregate truth. They propose that a user can be physically tracked by using the motion sensors in the phone and approximating the distance the person has moved.

While their argument has merit, the example does not. Assuming the application could determine a person has walked five blocks, that only tells the radius of the circle. The radius is also relative to an unknown location. Perhaps they have an absolute starting point, perhaps from pinging base stations and relying on Geo-IP systems. However, if they can an absolute value to represent the starting point, why not repeat that solution and avoid complex physics calculations?

## Android vs. iOS: The Security Battle (2014)

Apple’s iPhone runs on the iOS operating system and has taken a different approach to application security.

The first layer of defense is the iTunes store, that acts as a walled garden, by preventing the installation of any nontrusted application. An app becomes trusted through a verification process that is controlled by Apple, and then cryptographically signed by the developer. If an app is determined to be malicious, then Apple can simply revoke the developer’s certificate.

The next layers of protection use traditional desktop solutions such as Data Execution Protection (DEP) and Address Space Layer Randomization (ASLR). These protections increase the complexity to exploit software vulnerabilities by separating memory pages for data and code. Now that a memory corruption attack, such as trivial stack overflow, cannot directly execute its own payload it needs the memory address of system functions as a jump target. Under ASLR the address is randomized by the assembly loader and cannot be known in advance.

Android avoided these memory corruption attacks by requiring applications to follow the semantics of Java programming. Java does not allow direct access to memory, even through the Android Native Interface. Instead C++ implements are forced to marshal their allocations through Java wrappers.

# Conclusions

Mobile devices have become an integral part of modern life and as such, are targeted by malicious applications. The most popular operating system for these devices is called Android, an Open Source Solution (OSS). It protects the end user through a permissions system based on SE Linux.

While the operating system can reliably enforce policies there are numerous challenges to correctly represent those policies without arbitrarily growing the attack surface. Another concern comes from involving the end user in security critical decisions. These users lack the ability to understand the impact of these technical decisions. There have been multiple efforts to improve the user experience (UX), though it’s an uphill battle.

Alternatively, Role and Attribute Based Access Control (RBAC/ABAC) systems have been proposed to create distinct security contexts. These solutions have the capacity to greatly improve security on the device, however care must be taken to seamlessly integrate the multiple personas on a single device used by a single person. If this is not accomplished, then the user experience will suffer.

Aspect Oriented Programming (AOP) solutions, such as Figment, are likely headed the correct direction. By decoupling the business logic from the security policies, it becomes possible to run partially trusted code or take advantage of new solutions when they arrive. Perhaps, the third-party code should run remotely within a cloud provider’s ecosystem.

The truth is all solutions have elements of strengths and weaknesses, and its unlikely that one size fits all. Systems need to be adaptive to a gradient of paranoia and do nothing more.