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.

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

## 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.