Foreword

Chapter 1

Attacking Smartphone Security and Privacy

Vincent F. Taylor

Department of Computer Science, University of Oxford, Oxford, UK.

Prof. Ivan Martinovic

Department of Computer Science, University of Oxford, Oxford, UK.

CONTENTS

1.1 Introduction

The smartphone landscape continues to grow at an explosive pace as devices become more powerful, feature-rich, and more affordable to the average consumer. Gartner reports smartphone sales as exceeding 1.4 billion units in 2015, up 14.4% over the previous year [42]. Smartphones offer greatly increased functionality over traditional feature phones due to the availability of full-blown operating systems providing advanced APIs to third-party app developers. Smartphones are predominantly powered by Android or iOS, with Android maintaining a commanding lead of the market with 84.7% market share as of Q3 2015 with iOS in a far second at 13.1% [41]. Other operating systems represented in the landscape include Windows Phone and Blackberry among others. On top of the operating system, smartphones offer a variety of network interfaces for connectivity, multi-tasking facilities, and open APIs for supporting third-party app development. Android and iOS have rich app marketplaces each offering access to approximately 1.5 million apps [105, 104] that add additional functionality to the smartphone. The always-connected, extremely extensible nature of smartphones, exposes a large footprint on the device where weaknesses in the underlying hardware or software may be exploited by an attacker.

The smartphone landscape is very large, and has a number of layers of software, protocols, and services that work together to deliver an experience to the consumer. The interaction between consumer, apps, smartphone, service provider, and the wider internet is supported by various wireless protocols that provide connectivity. Thus, a smartphone may be vulnerable to attacks coming from installed apps, wireless interfaces, running services, and the underlying configuration of the device. We are motivated to systematize this knowledge of attacks and attack vectors, as this will provide a compendium to security researchers intending to develop intrusion detection and prevention systems[1] for the smartphone ecosystem. We do this by comprehensively enumerating the ways in which the security and privacy of a smartphone can be attacked. By understanding the ways in which smartphones can be attacked, we obtain a mechanism to compare them to traditional workstations, giving useful insight into the additional or varied risks that need to be addressed when building technology to secure smartphones.

[1] For brevity, we refer to intrusion detection and prevention systems as simply IDS for the remainder of this chapter.

1.2 Background

To understand the ways smartphones are attacked, we first need to understand the operating systems that run on these devices and the security models and features they employ. There are four major smartphone operating systems: Android, iOS, Windows Phone, and Blackberry [111]. In addition to typical low-level tasks such as memory management and process scheduling, smartphone operating systems provide features critical in today's smartphone landscape, such as allowing access to a touchscreen, camera, Bluetooth, Wi-Fi, NFC, GPS, microphone, and other such hardware. Aside from providing access to the typical smartphone hardware, the operating system also mediates access to the underlying cellular radio, enabling communication with a mobile network carrier.

1.2.1 iOS

iOS is a mobile operating system developed by Apple Inc. It has a healthy app ecosystem that surrounds it with over 1.4 million iOS applications available for download. The operating system itself is proprietary, closed source, and written in C, C++, Objective-C and Swift. It is a Unix-like operating system and features a hybrid kernel that runs on 64/32-bit ARM processors. Before iOS apps are made available to the public in the Apple App Store, they must undergo a thorough vetting process by Apple. Apps must pass reliability testing and other analysis to ensure that they are not malicious or otherwise unsavoury. Apple's vetting process includes manual testing and static analysis to determine whether an app tries to perform actions outside of what it claims to do [3]. This vetting process is not always perfect and indeed security researchers have uncovered ways of circumventing the protections put in place by Apple [43]. In the case of Jekyll [113], the malicious app passed the vetting process by rearranging its code to add new, malicious functionality, after passing the approval process. The iOS kernel uses code signing to ensure that all apps running on a device come from an approved source and have not been tampered with [4]. Additionally, all third-party apps are sandboxed by iOS to prevent them from accessing data stored by other apps and modifying the system. However, Han et al. described how to 'break out' of the iOS sandbox by leveraging dynamically loaded, private APIs in malicious apps [50]. Finally, iOS enforces a secure boot chain and file encryption using a per-file key.

1.2.2 Android

Android is a mobile operating system developed by Google and the Open Handset Alliance. Android devices are powered by a healthy app ecosystem providing access to over 1.6 million apps. The core of the operating system is written in C, with additional components written in C++, and the user interface portions written in Java. Like iOS, it is also a Unix-like operating system, however, it features a monolithic kernel, designed to run on a number of processor platforms such as ARM, x86, and MIPS. In stark contrast to the Apple App Store, the Google Play

Table 1.1: Summary of the main characteristics of Android, iOS, Windows Phone, and Blackberry that contribute to their attack surface.

Operating System	Android	iOS	Windows Phone	Blackberry
OS Family	Linux	Darwin	Windows CE-7, Windows NT-8	QNX
Vendor	Google Inc., Open Handset Alliance (and OEMs)	Apple Inc.	Microsoft (and OEMs)	Blackberry Ltd.
CPU Architecture	ARM, ARM64, x86, MIPS	ARM, ARM64	ARM (ARM64 upcoming [70])	ARM
Market Share (2015 Q3) [41]	84.7%	13.1%	1.7%	0.3%
Source Code	Open	Closed	Closed	Closed
Programming Language	C, C++, Java	C, C++, Objective-C, Swift	C, C++	C++
Application Store	Google Play	App Store	Windows Phone Store	Blackberry World
Reverse Engineering Tools	apktool, dex2jar, JD-Compiler, XDA auto tool	iRET toolkit, Windows Explorer, oTool, iExplorer, Class-dump-z	Decompresser, Visual Studio, .NET Decompiler	JD-GUI, VSMTool, COD extractor

Store does not require an exhaustive app vetting process before an app is admitted to the store. In general, apps are dynamically tested with a Google security service known as Bouncer [52]. Google automatically scans apps using dynamic analysis and combines the results of this analysis with signals from its reputation engine after it has analysed the account of the app developer themselves. Security researchers John Oberheide and Charlie Miller demonstrated techniques that could be used to fingerprint Bouncer [84]. They identified unique characteristics of the Bouncer emulation framework such as the hostname, phone number, and Android ID. By checking for these fingerprints, malware can pretend to be benign when being tested by Bouncer and then become malicious when installed on victim devices. In an early study [37], Enck et al. analysed the source code of 1100 Android apps and found no evidence of malware or exploitable vulnerabilities. Unfortunately, the landscape has deteriorated since then. Indeed, Zhou and Jiang [121] provide a characterisation of the evolution of Android malware. On the Android platform, every app runs in its own sandbox by default. As a result of this, each app is isolated from other apps and the system itself, except by using well-defined APIs and system services such as inter-process communication (IPC). However, researchers have found ways for apps to break out of their sandbox and read arbitrary files using symbolic links [76]. Android uses a Linux-like user approach, where each app is executed as a different user and thus inherits the security provided by the operating system in protecting its resources and files. In addition to sandboxing and permissions, Android is also designed to prevent platform modification by malware and also has the capability of remotely removing malware from a device if required [65].

Comparison of Smartphone Operating Systems. Table 1.1 shows a comparison of the similarities and differences between the four most popular smartphone operating systems, and summarises our effort in distilling this information from the literature [8, 56, 86, 26, 100, 82]. For brevity, we do not compare an exhaustive set of features for these operating systems. Instead, we target the main characteristics of the operating systems that contribute the most to vulnerabilities, and thus are most interesting to IDS developers. We look at the OS family, CPU architectures supported, source code model, programming languages used, and reverse engineering tools that are available. Android is based on the Linux-family of operating system, while iOS' Darwin and Blackberry's QNX are Unix-like operating systems. The outlier here is the Windows Phone operating system which is built around the Windows family of operating systems. All four of these smartphone operating systems are built to run on ARM processors, with Android offering the capability to run on x86 and MIPS processors as well. Android dominates the market, being delivered on 82.8% of smartphones, iOS on 13.9%, and Windows Phone and Blackberry trailing distantly with 2.6% and 0.3% deployment respectively, as of 2015 Q3 [41]. Android is the only open-source operating system on the list and all are written in C/C++ or other variants of C. Each of the operating systems is supported by a single official application marketplace which provides third-party apps to users. Importantly, Android and Windows Phones have OEMs that (sometimes) modify the standard operating systems and unwittingly introduce vulnerabilities [114]. Finally, all four operating

systems have a suite of reverse engineering tools available to assist with vulnerability analysis.

Definitions: We now define key terms that are used throughout the chapter.

- Attack vector: The means by which an attack is carried out against a system.
- Exploit: The method used to take advantage of a vulnerability.
- Vulnerability: Any weakness in a system that exposes it to risk.

Threat Model: In evaluating the landscape as it concerns intrusion detection and prevention, we systematize adversaries based on their capabilities, goals, and relationship to the smartphone under attack:

- *Local adversary (active/passive)* - This attacker is present on or controls the local network. A local attacker may also be logically adjacent to the device (for example, spoofing a cell tower) or have close physical access to a device (for example, in close physical contact with the victim).
- *Remote adversary (active/passive)* - This attacker is present outside of the local network and may control segments of the network between the victim and the destination of their traffic.

Passive adversaries may eavesdrop on and observe traffic from the communication channels in the network. They may also observe data from *side channels*, such as device sensors [13], power consumption [117][57], and wireless transmissions [106][20]. Conversely, active adversaries are able to read, modify, or inject data into a communication channel. Note that malicious app developers (or adversaries who modify/repackage apps) fall into the category of active remote adversary. Our types of adversary are not necessarily mutually exclusive. Indeed, adversaries may change position in the network and more than one adversaries may collude to achieve a more complex objective. The specific target of the adversary may be one or more of:

- *The victim themselves*: The adversary is intending to cause harm to the victim and does this by attacking their smartphone to cause loss of data or perform denial-of-service.
- *The device itself*: The adversary may be intent on exfiltrating personal data from the device such as contacts, credit card information, social security numbers, pictures or videos. In the case of corporate espionage, the adversary may be targeting the employee of a company to obtain intellectual property, unpublished reports, or other sensitive business data.
- *Device resources*: Data on a device may be immaterial to an adversary who is targeting smartphones to exploit their resources such as storage, processing power, and bandwidth. This is especially common for adversaries interested in 'recruiting' devices for a botnet.

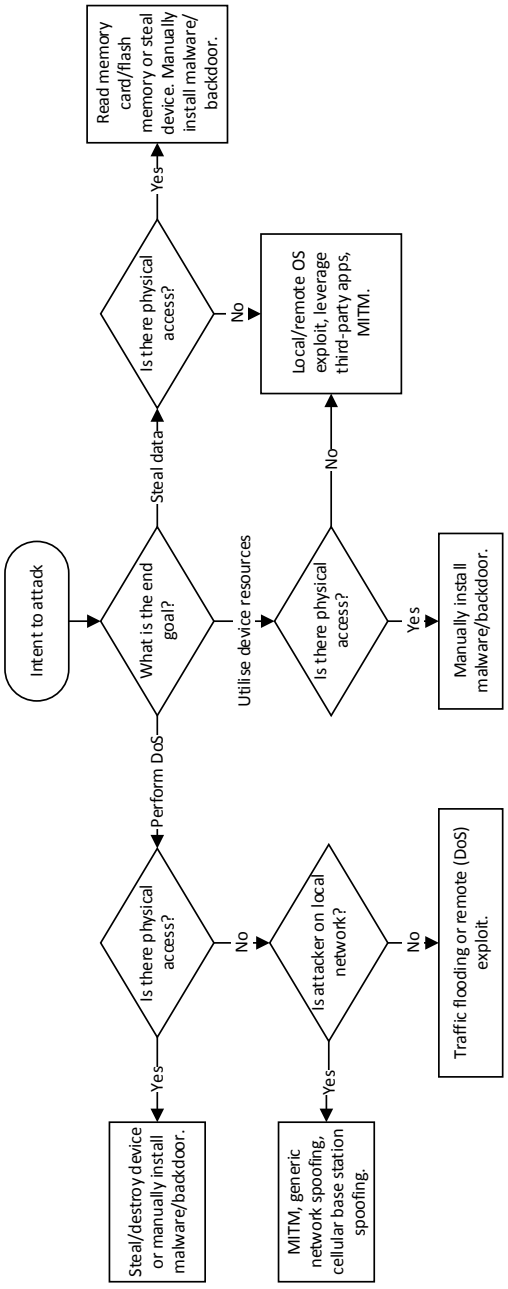

Figure 1.1: Flowchart of a typical smartphone attack from the attacker's point of view.

For the remainder of this chapter, we frame the attacks and attack vectors in relation to the position and intent of the adversary. This is summarised by the flowchart of the decision-making process of an attacker shown in Fig. 1.1. In general, an adversary has one of three objectives when attacking a smartphone:

- *Perform DoS*: The adversary is concerned with preventing the device from performing its prescribed functionality. This attack is fairly noticeable, since the user will perceive degradation in performance or a missing device (in the case of theft).
- *Utilise device resources*: The adversary is concerned with leveraging the resources (CPU, memory, network access) of a device to further their own goals e.g. recruiting devices for a botnet or as a proxy for launching further attacks.
- *Steal data*: The adversary is concerned with obtaining sensitive data from a device such as user account information, credit cards, multimedia, and sensor data. Note that the sensitive data the adversary is interested in may not yet exist, so the adversary may plant a backdoor, e.g. when spying on a spouse.

1.3 Smartphone Attack Vectors

Developing IDSs for smartphones is complicated by the fact that smartphones are devices that communicate over a variety of wireless interfaces/networks and provide a highly customisable and extensible platform. Thus, smartphones will necessarily have a number of areas that must be exposed in order for them to provide their stated functionality. Moreover, what really distinguishes smartphones from other computing platforms is the multitude of sensors they contain and their ultra-high mobility which makes them susceptible to loss/theft/physical access. The following list distils [40, 33, 72] 13 vulnerable areas (or 'weak-points') on typical smartphones that will continue to be targets for delivering exploits:

- **Browser**: May contain vulnerabilities in parsing webpages, processing Javascript, or providing WebView functionality to apps.
- **Baseband processor**: Smartphones can be tricked into connecting to rogue base stations which can then attack the mobile radio interface.
- **Messaging services**: SMS/MMS messages may be used to deliver malicious payloads.
- **Wireless interfaces**: Attackers can attempt to attack a smartphone from any one of the myriad of (non-cellular) wireless interfaces.
- **SIM card**: Attackers may be able to manipulate SIM cards to attack a device or steal data.

- **Memory card**: Many smartphones provide slots for external memory cards. These are frequently unencrypted and data can be retrieved if the smartphone or memory card itself is misplaced.
- **Hardware interfaces/ports**: Smartphones may be vulnerable to attacks coming over their exposed ports, such as USB ports. By opening a device, an attacker can also try to pilfer data through low-level circuitry such as JTAG ports.
- **Operating system**: Adversaries can attack typical weaknesses found in operating systems. In some cases, smartphone operating systems may not be as mature (or robust) as desktop operating systems.
- **Third-party apps**: Third party apps can access any resource that they have been given permission to access. Additionally, apps can attempt to break out of the OS-provided sandbox. Attackers can also use vulnerable third-party apps as a proxy for conducting further attacks on a smartphone through privilege escalation.
- **Users**: Users can advance attacks if they make bad device configuration choices or are victims of social engineering.
- **Memory**: Physical memory on the device can be modified to remove protective mechanisms from the system.
- **Firmware**: An attacker may target sub-module (such as Wi-Fi interface cards) firmware to obtain long-term elevated privileges on the device.
- **Device itself**: An attacker may target any one of the side channels coming from the device itself, for the purposes of device fingerprinting or recovering sensitive data such as encryption keys or screen-lock codes.

We systematize attack vectors as belonging to either of four categories: drive-by attacks, app ecosystems, physical attacks, and social engineering [79, 95, 40]. In detailing attack vectors, we use *italicized text* to denote the vulnerable areas affected. Also, we only briefly identify some attacks to put the attack vectors into perspective. We leave a detailed treatment of all the major attacks against smartphones for Section 1.4.

1.3.1 Drive-by Attacks

Vulnerable areas affected: Browser, baseband processor, messaging services, wireless interfaces, operating system, third-party apps, users.

In the case of drive-by attacks (or watering hole attacks), an attacker attempts to exploit existing bugs in the software running on the smartphone that processes external

data. A common method of delivering drive-by attacks involves exploiting vulnerabilities in the *browser* on the smartphone to make it execute a malicious payload. These attacks can be carried out *en masse* since popular web pages can be compromised and laced with malicious payloads. Alternately, links to malicious pages can be sent to *users* through traditional channels such as email, *messaging services* and social media. An attacker can also manipulate unencrypted HTTP pages to insert malicious payloads or take advantage of improperly handled SSL/TLS (in *third-party apps*) to perform a MITM attack [38] and insert the malicious payload that way. Note, however, that attacks targeting smartphone browsers have more limited potential than desktop browsers due to application isolation via sandboxing.

Drive-by attacks can also leverage any one of a device's *wireless interfaces*. Various spoofing attacks can be performed against Wi-Fi and Bluetooth, and indeed, base station spoofing can used against the *baseband processor*. Other targets of drive-by attacks may include WebViews (*operating system*), a user interface component in smartphone development frameworks that allows apps to easily render webpages from within the app. A WebView can be used to provide an interface to a Javascript component and thus external Javascript can be executed on a device. Additionally, WebViews can also allow a website to access data stored on devices. If an attacker is able to intercept or modify the content of a URL that is loaded by a WebView (using a MITM attack or Cross-Site Scripting (XSS) for example), they can use functions from within the WebView framework to access data from the device. Worryingly, privilege escalation exploits have been published that allow arbitrary code execution on vulnerable devices through WebViews [91]. Another class of attacks, known as component hijacking attacks [67], leverage the drive-by attack principle to access private data and spoof intents.

Drive-by attacks may exploit network services, pieces of software running on a device that open ports to listen for incoming connections. Traditionally, network services only run on devices acting as servers, such as web (HTTP) servers listening on port 80. In the smartphone landscape, however, to satisfy the great need for interconnectivity, mobile devices can be found running network services such as ADB [112], VPN, VNC, RDP and SSH services. In the case where smartphones are configured to share their internet connection through a mobile hotspot, they can be expected to also run DHCP services and act as a default gateway. These additional services all increase the number of avenues for exploit. Network services offer an attractive interface for attackers to attempt to exploit, since they provide a (usually) always open entrance that is accessible via the network. Exploiting network services is also particularly attractive to an adversary since no user intervention is typically required to allow the exploit to take place and after successful exploitation there may be no immediate indication to the user that an attack has indeed happened. By default, smartphones may not have any network services installed, but there are a wide variety of third-party apps that users install which offer additional functionality that requires the use of network services. Indeed, Nielson reports that the average user uses 26.8 different apps per month [81], and any of these could potentially leverage network services.

Drive-by attacks, while successful on traditional workstations, may have more

limited impact when translated to the smartphone arena. On Android, most software is implemented in Java and executed by the Dalvik Virtual Machine. This mitigates some of the typical attack strategies (such as buffer overflows) since low-level data structures are protected by boundary checks. However, many Android apps also leverage libraries implemented in native code; thus some parts of many apps continue to be susceptible to traditional attacks against memory corruption bugs. These attacks are quite dangerous since they can lead to code execution on the device [48], with the user not necessarily knowing that they have been compromised.

1.3.2 App Ecosystems

Vulnerable areas affected: Third-party apps.

By and large, user-installation of grayware/malware is limited due to the use of 'trusted' software repositories such as the official app stores. App stores and smartphone operating systems utilise strong technical mechanisms to ensure a restriction on the *third-party apps* that can be installed on a device. Attacks coming from app ecosystems leverage the fact that if grayware/malware can be placed in a marketplace, it can quite quickly be available for infecting the entire ecosystem. Additionally, grayware/malware authors are incentivised by the fact that users are typically more trusting of apps if they find them in the official app marketplaces.

As mentioned in Section 1.2, app stores employ various degrees of vetting before allowing an app to become available for the general public to download. Additionally, smartphone operating systems restrict, by default, the 'sideloading' of apps, i.e., installing apps to a device through unofficial channels. For this reason, most grayware/malware affecting smartphones are delivered as Trojan-horse apps via an app store. Thus malicious authors must develop apps with some functionality, but containing malicious payloads hidden from app store vetting using timebombs, dynamic code loading, reflection, code obfuscation and/or IP address checking (to determine whether the app is being run through an app store's vetting engine). Recently, the BrainTest trojan [1] utilised all the aforementioned strategies to evade app store detection.

One strategy used by grayware/malware authors, and most commonly observed in third-party app marketplaces, involves the repackaging of legitimate apps to inject malware [121] which can then attempt to exploit the *operating system/firmware* or steal data from the *memory card*. Fraudsters have also been known to modify the advertising portions of legitimate apps to insert their own code. This allows them to fraudulently obtain revenue from a legitimate app [120]. Other less malicious apps (and their included libraries) have been known to leverage additional and unnecessary dangerous permissions, ostensibly to have greater access to sensitive data and resources, which can then be used for profiling a user [16, 122, 39] for reasons such as better advertisement targeting, or more maliciously, selling user data directly to other third-parties.

Smartphone worms are much more limited than Trojan-horse apps but may begin to see wider adoption with the availability of operating system exploits propa-

gated by modern smartphone connectivity features such as portable hotspots/NFC and even older channels such as Bluetooth/SMS/MMS/WAP. Operating system protection mechanisms, such as SELinux, offer mitigation for system exploits by enhancing the boundaries of app sandboxes [2] and thus worms may have more limited success.

1.3.3 Physical Attacks

Vulnerable areas affected: SIM card, memory card, hardware interfaces/ports, memory, firmware, device itself.

One class of physical attacks come about from dismantling the *device itself* and/or being able connect to and interface directly with the *hardware interfaces/ports* on the device; we call these *physical (tampering)* attacks. Physical attacks may also make use of side channels that enable the inference of private data located on a device; we call these *physical (general)* attacks. We expand on specific physical attacks in Section 1.4.1, but right now we enumerate general physical attack approaches:

1. Accessing a device that does not use a screen-lock and transferring the data from the device using copy/paste/attach features within the operating system.
2. Accessing *memory cards* within the device itself and removing them to obtain data that was stored on the device.
3. Inferring PIN/screen-lock codes from smudges on a smartphone touchscreen [7].
4. Leveraging ports, such as USB ports, on the device to perform further attacks [61, 59, 69].
5. Modifying physical *memory* chips on the circuit board to introduce new software and/or affect the *firmware* of low-level hardware (such as Wi-Fi adaptors).
6. The *SIM card(s)* in a device can be removed to retrieve sensitive data such as messages and phone numbers. Malicious payloads can also be written to a SIM card.

An attacker can leverage their access to the *hardware interfaces/ports* of a device to place malware or other data on the device or execute commands. Lau et al. demonstrated how it was possible to install arbitrary apps on an iOS device through the USB port [61]. The Android Debug Bridge (ADB) can also be used to launch attacks. The ADB is a command line tool that can be used to connect to and run commands on Android devices using a desktop.

Another class of physical attacks comes from leveraging the physical state of a device or physical access to the device to attack it. Leveraging the physical sensors on a smartphone is an example of utilising the physical state of a device to enable

attacks. The literature exemplifies using the accelerometer/gyroscope [13, 116], and light sensor [103] to steal device credentials/passwords.

Attackers can also leverage physical access to a device to attempt to pass the 'lock screen', provided that screen-locking is enabled in the first place [36]. A locked device is usually guarded by PINs, patterns, and passwords, and more recently, using biometrics such as fingerprints. An adversary being able to successfully unlock a device depends on the complexity of the credential used to lock the device. Approaches as rudimentary as looking at screen smudges have demonstrated potential in assisting attackers to bypass locked screens [7]. Biometric approaches, which show much promise, have been shown to be dangerous if implemented incorrectly [118], with the end result a potential compromise of a user's biometric data such as a fingerprint. Needless to say, the compromise of a user's biometric is a serious problem, as by nature it cannot be replaced.

1.3.4 Social Engineering

Vulnerable areas affected: Browser, operating system, users.

With social engineering, the user of a smartphone is tricked into revealing credentials or performing actions that assist the attacker in furthering their attack. These attacks are dangerous in that they employ non-technical strategies to elicit private information from users and, as such, generic IDS solutions to address social engineering are not available. The problem of social engineering is exacerbated by the fact that users may not know that they have been successfully attacked until long after the fact, if at all. Three common social engineering attacks specific to smartphones are:

1. *Making malicious apps look like legitimate apps.* Malware/grayware authors typically build clones of popular applications to trick a user into installing their version because it has a name and description very similar to the app they actually want [45][120].

2. *Enticing users using device-specific details.* Smartphone users may be tricked by ads and webpages that give them advice specific to their device make and model. Attackers commonly use the User-Agent sent by a browser/app to identify the device before sending customised messages to the user about faults with their specific device such as poor battery or malware infections. The users are then led to download malware which supposedly solves their 'problems' [49].

3. *Malware pretending to be a second factor of authentication.* Desktops infected by the Zeus malware may instruct users to download an authentication component to their smartphone as a second factor of authentication when the user attempts to log in to their online bank [18]. The malware then captures a user's bank login credentials.

Aside from social engineering that leads to malware installation, other typical social engineering attacks that result in the user giving away their credentials are just

Table 1.2: Table showing attack vectors and what vulnerable area on the smartphone they target.

Attack Vector	Vulnerable Area Affected
Drive-by attacks	Browser, messaging services, wireless interfaces, SIM card, memory card, operating system, third-party apps, users, memory, firmware
App ecosystems	Third-party apps
Physical attacks	Baseband processor, SIM card, memory card, hardware interfaces, USB, memory, firmware
Social engineering	Browser, operating system, users

as detrimental as on traditional desktops. Especially considering that a user may be logged into several services from their smartphone at the same time, social engineering presents a high-reward attack vector to adversaries.

Table 1.2 summarizes the relationship between the attack vectors and the vulnerable areas that they target. From the table, it can be seen that drive-by attacks have the potential to affect the most areas on a smartphone. This is perhaps unsurprising, as drive-by attacks are made possible by bugs/vulnerabilities in the software on the smartphone itself, and thus there is a rich attack surface that can be targeted. Physical attacks have the second largest number of vulnerable areas and target weaknesses in the physical hardware/characteristics of the smartphone. App ecosystems and social engineering target fewer vulnerable areas directly, but can be used as a proxy for delivering more dangerous drive-by exploits if users are tricked into installing apps or performing particular actions on their device.

Table 1.3 shows common attack vectors, the level of sophistication required to achieve success, and the potential effect of device compromise. The level of sophistication refers to the technical expertise required from the attacker and ranges from Low (minimal technical ability required), Medium (moderate technical ability required, with published exploits easily available/adaptable), to High (advanced technical ability required, usually requiring the development of zero-day exploits or advanced reverse-engineering skills). Effect of compromise ranges from Low (information disclosure or minor annoyance), Medium (Low + greater annoyance and potentially costing the user money e.g. premium rate SMS/calls), to High (full compromise of the device with unfettered access by the adversary). The main insight from Table 1.3 is that social engineering requires minimal skill and has the potential to affect many victims, but the effect of the attack is typically low. Drive-by attacks, on the other hand, can prove very effective to attackers since published exploits are available (especially for older devices which are still widely used [110]) and can yield good returns in terms of effect of compromise while targeting a moderate number of victims. Physical tampering of devices requires high sophistication by adversaries but may yield significant rewards and are usually employed at the nation-state/law-enforcement level. Worryingly, unsophisticated attackers can com-

Table 1.3: Table showing attack vectors and their main characteristics.

Attack Vector	Level of Sophistication	Effect of Compromise
Drive-by attacks	Medium/High	Low/Medium/High
App ecosystems	Low	Low/Medium
Physical (tampering)	High	High
Physical (general)	Low/Medium	Low/Medium
Social engineering	Low	Low/Medium

bine social engineering with published drive-by exploits to obtain a significant return on investment, especially if attacks target users with older devices.

1.4 Smartphone Attack Hierarchy

We classify smartphone attacks based on the position of the attacker in the 'space' relative to the smartphone under attack as follows:

- **Physical vs. Non-physical:** As shown in Fig. 1.2, the first level of differentiation is whether the attack is performed by physically accessing the device. This is a logical separation of attacks as it broadly divides attacks into those which require tangible access to a device as opposed to those that access the device in an intangible way. IDS developers will typically focus on non-physical attacks. Non-physical (or intangible) attacks can be further separated into two categories: local and remote.
- **Local vs. Remote:** Local and remote refer to the logical proximity of the attacker to the victim device in terms of location on the network. Broadly speaking, local attacks are carried out by attackers that are on the current local area network (LAN) segment (or otherwise logically adjacent to the victim) and includes well-known attacks such as ARP spoofing, MITM, and traffic analysis attacks. Remote attacks are carried out by adversaries who are able to launch attacks from beyond the local network segment, i.e., more than one network hop away.
- **Interactive vs. Non-interactive:** Attacks can also be categorised into whether they are interactive or non-interactive. By interactive and non-interactive, we mean whether the smartphone user is required to perform a particular action on their smartphone for the attack to be successful. In general, non-interactive attacks are more attractive (and more difficult to exploit) since no user intervention is required and, as a result, may be more stealthy.

In Sections 1.4.1 and 1.4.2, we compare typical attacks that fall into the categories of physical and non-physical, to assess their characteristics relative to each other and

gain an understanding of the motivations of attackers for using one type of attack over another. Table 1.4 provides a compendium of examples of attacks for all the exploits mentioned in this section.

1.4.1 Physical Attacks

Physical attacks are carried out by attackers that target the hardware of the device itself. In other words, these attacks require attackers to physically touch the device in order to carry out their malfeasance. The main classes of physical attacks are: hardware tampering, attacking the device over its built-in ports, and leveraging physical sensors on the device to garner data.

Hardware Tampering. Hardware tampering attacks are directed at the physical circuitry of the device itself. Security of hardware is often an afterthought since many manufacturers consider the device hardware secure through obscurity. Tampering with hardware requires esoteric knowledge and a particular skillset, but common low-level interfaces and circuitry on most electronic devices allow attacks to be performed against a wide range of devices. For example, many electronic devices, when disassembled to the circuit board level, will have exposed serial and JTAG ports. These ports can be used to intercept debug messages, send commands, or flash the firmware of the device. Serial and JTAG interfaces are widely used for communication between sub-modules in embedded systems and an attacker with reasonable skill and patience can usually find ways of accessing these buses. By being in physical possession of or in close proximity to a device, an attacker may also leverage data gleaned from any of the physical side channels on the device such as power consumption or electromagnetic emanations. By leveraging side channel information, attackers can cheaply [47] determine secret keys from a device's embedded circuitry [57].

Even if the attacker cannot leverage the aforementioned interfaces in a reasonable way, they can attack other components of the device such as the flash memory. The attacker can potentially de-solder flash memory from the circuit board, and use other tools and hardware to read and modify the filesystem, bootloader, or other sensitive configuration or software [51][102]. Munro [74] showed how to leverage hardware tampering to read data from a device, crack PIN codes, boot a smartphone from an operating system installed on a memory card, read from and write to a UART, access flash memory, and wipe a device.

Ports. Smartphones can also be attacked using their built-in ports. Of all the physical ports on a device, the standard USB port is most commonly used. The attack to be launched at a device over USB depends on the USB mode that the device is in. Common USB modes include mass storage, media device, tethering, fastboot, and ADB. Devices running iOS were shown to be vulnerable to arbitrary app installation over USB [61]. Android devices that have USB debugging enabled have the ADB daemon running on the device. The ADB daemon allows the running of commands with special system privileges. Using ADB, an attacker can bypass some of the security

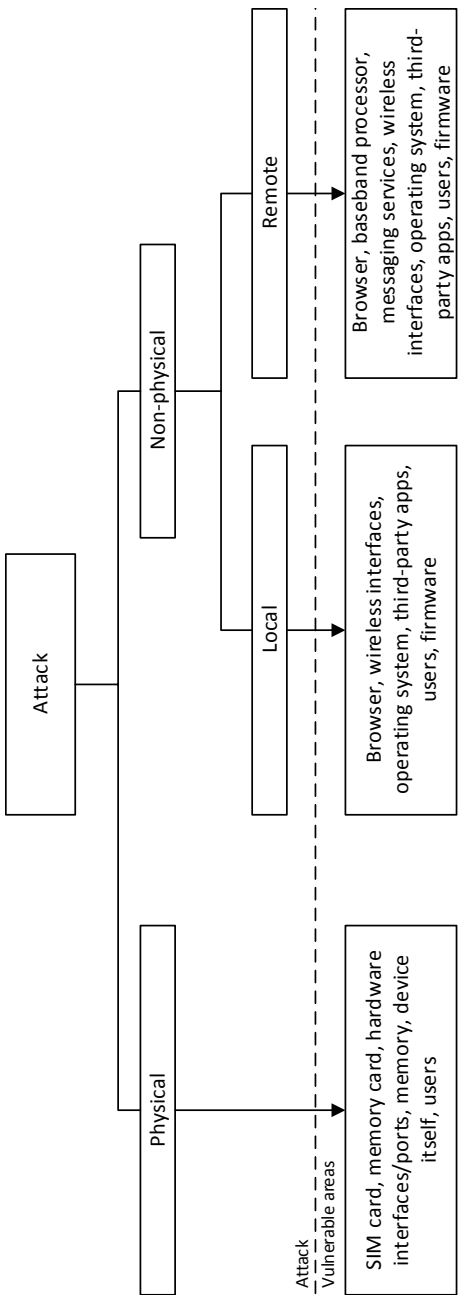

Figure 1.2: Taxonomy of smartphone attacks.

features of the operating system. It is important to note, however, that the majority of Android devices will not have ADB enabled by default (since it is a feature mostly used by developers) and later (>= 4.2.2) Android devices have key-based authentication of desktops. However, vulnerabilities found in Android 4.2.2-4.4.2 allowed attackers to bypass ADB authentication [77]. An interesting avenue for attacking a locked ADB-enabled device is to obtain access to a desktop that is already permitted to connect to the device, and using ADB to execute commands on the device via a terminal on that desktop. It is expected that only few devices will be vulnerable to attacks using ADB, however, the attractiveness of this approach is increased when considering the power gained from attacking a device that is ADB-accessible.

In a popular attack called 'juice-jacking', attackers can steal a victim's data or install software on their device via the USB port [59]. USB ports on a smartphone are not the only ports that are vulnerable. Indeed, other physical ports that are ripe for attack are the SIM card ports, SD card ports, HDMI ports and docking connectors, to name a few.

Physical sensors. Smartphones have a plethora of built-in sensors that enable rich interactions between users and apps. By leveraging sensors such as the accelerometer/gyroscope, researchers showed how it was possible to infer keystrokes on a smartphone based on how the device moved as credentials were being typed in [13, 116]. Other researchers showed that minor movements when entering credentials are sufficient to change the ambient light hitting the light sensor in a way that can be exploited to reduce the search space required for guessing a PIN [103]. The microphone on a smartphone has also been used to extract high-value audio data from smartphones [94]. Using power side-channels, researchers were able to clone SIM cards [117] and retrieve cryptographic keys [57]. Other researchers used imperfections in smartphone sensors for device fingerprinting and tracking [30]. One of the big differentiators between smartphones and traditional computers is the myriad of built-in sensors that they contain. Thus sensor side-channels contribute a novel attack surface to smartphones and is an open area for further IDS research.

1.4.2 Non-Physical Attacks

Non-physical attacks on a smartphone do not require physical manipulation of the device. Thus these attacks will predominantly come from across a communication channel, be it Wi-Fi, 34/4G, Bluetooth, etc. As mentioned earlier, local attacks come from the local network segment from physically (or logically) adjacent devices. These attacks are predominantly launched by attackers on the same Wi-Fi network as the victim. Remote attacks come from attackers further removed than the local network segment, i.e., more than one hop away. Remote attacks can come from hosts on the internet and may be delivered by malicious web pages, email, or by exploiting apps installed on a device. Potential attack vectors also include malicious SMS/MMS messages carefully crafted in a way that exploits a vulnerability on the system [80]. Some attacks can also be categorised as being 'local or remote' in cases where they

Table 1.4: Examples of common attacks against smartphones and their characteristics.

Name	Vector	Vulnerable Area	Local/Remote	Interactivity	Examples
Man-in-the-middle attacks	Drive-by	Browser, third-party apps, baseband processor, wireless interfaces	Both	Non-interactive	SSL MITM on apps [38][54]
Network spoofing	Drive-by	Wireless interfaces, browser, operating system	Local	Non-interactive	DNS spoofing [89], DHCP spoofing [35], ARP spoofing [9][101], Wi-Fi attacks [89][32]
Base station spoofing	Drive-by	Baseband processor, messaging services	Local	Non-interactive	Cell tower spoofing [93]
Network service attacks	Drive-by	Operating system, network services	Both	Non-interactive	ADB exploits [112], SSH exploits [96]
SMS/MMS/WAP attacks	Drive-by	Messaging services	Remote	Both	Stagefright [80][119][123], iOS SMS bugs [46], sending USSD codes via WAP [60]
Software attacks	Drive-by, app ecosystem	Browser, operating system, third-party apps	Remote	Interactive	iOS browser weaknesses [34][73], WebView attacks [68], Jekyll App Store vetting evasion [113], integer overflow attack [48], bypassing sandbox [76], malicious iOS apps [50], malicious Android apps [1]
NFC attacks	Physical	Wireless interfaces	Local	Both	Android NFC exploit [75] [71]
OS attacks	Drive-by	Operating system, third-party apps	Remote	Non-interactive	Generic iOS vulnerabilities [22], generic Android vulnerabilities [24], generic Windows Mobile vulnerabilities [25], generic Blackberry vulnerabilities [23], GTalkService exploits [85][83], LibTIFF Buffer Overflow [92]

Name	Vector	Vulnerable Area	Local/Remote	Interactivity	Examples
Bluetooth attacks	Drive-by	Wireless interfaces	Local	Both	Bluejacking, blusnarfing, bluebugging [108][62][14]
Physical attacks	Physical	Hardware interfaces/ports, memory, memory card, firmware, SIM card, device itself	Local	Non-interactive	Android forensics and scraping [74][51][102], theft, cloning SIM cards [117], deriving secret keys [57]
USB attacks	Physical, drive-by	Hardware interfaces/ports	Local	Non-interactive	Juice Jacking [59], Juice Filming [69], Mactans: installing apps silently [61]
Side-channel attacks	Physical, drive-by	Device itself, wireless interfaces, SIM card, users	Local	Non-interactive	Inferring app usage [20], device fingerprinting [106][19][107], deriving secret keys [57], cloning SIM cards [117], detecting installed apps [27], inferring app usage patterns [20], inferring user mood [63]
Social engineering attacks	Social engineering	Users, third-party apps	Both	Interactive	App cloning [45][120], arousing curiosity [49], Trojan-horse attack [18]
Sensor attacks	Physical	Device itself	Both	Non-interactive	Leveraging accelerometer/gyroscope [13], using light sensor [103], using microphone [94]
Authentication attacks	Physical	Device itself, users	Local	Non-interactive	Smudge attack [7], attacks against biometric authentication [118]

can be launched from either location.

Local. On the local network segment, smartphones are susceptible to the typical spoofing attacks that traditional workstations are also susceptible to. With ARP spoofing [9][101], the attacker convinces the victim device that it is the gateway. The same principle applies for DNS [89] and DHCP spoofing [35] attacks. In DNS spoofing, the attacker responds to the victim's DNS queries. Usually the responses point a domain name to an IP address under the control of the attacker. By doing this, the attacker receives traffic from the victim that was intended for a different recipient. Similarly, with DHCP spoofing, the attacker responds with false DHCP responses, usually telling the victim to use the attacker's IP address as a default gateway. Once again, this tricks the victim into sending their traffic to the attacker. Once the attacker receives the traffic, they are free to inspect, modify, and/or forward the packets to some destination.

Alternately, the attacker can simply drop the packets and cause a denial-of-service (DoS) attack. It is worth noting that the attacker need not be present on the local area network as a client to perform local attacks. The attacker can use additional hardware such as femtocells or wireless access points to trick the victim into connecting to cellular [87] and Wi-Fi networks directly under their control [93]. If successful, the attacker has full domain over all traffic from the victim and is free to manipulate it as they see fit. The level of effort and sophistication required to carry out attacks over cellular networks is currently high but falling rapidly due to the availability of picocells/femtocells and open-source cellular infrastructure software [87]. Picocells can typically supply coverage within a range of 200 metres and can thus target thousands of potential victims if placed in a crowded area. By spoofing the cellular network, attackers essentially become a man-in-the-middle and can attack devices either by sending low-level commands, or perform more traditional attacks against internet connectivity like a typical man-in-the-middle. Once the MITM attack is underway, the attacker can potentially insert malicious payloads into the content being received by users using their choice of drive-by attack.

Local or Remote. Certain attacks can be carried out whether the attacker is on the local network segment or somewhere on the path to the destination. Two notable examples of attacks in this category are traffic analysis attacks and MITM attacks.

- ***Traffic Analysis***: By doing traffic analysis, an attacker is able to glean additional information from network traffic. Traffic analysis is usually done on encrypted traffic since the payload cannot be examined directly. By leveraging pattern matching or statistical analysis on captured traffic, the attacker can determine things such as the apps that a user has installed on their device [107][27], specific actions that the user is doing within these apps [20][19], or identify devices on the network based on their unique traffic fingerprint [106].

- ***MITM***: If an attacker successfully launches a MITM attack against a victim, they have the power to intercept, analyse, modify, and forward the traffic as they see fit. If transport layer encryption, such as TLS, is used, an attacker may be limited in what they can do with the traffic. Assuming that the victim properly validates and handles TLS certificates, the attacker would no longer be able to read or modify (without detection) the victim's traffic. However, the attacker would still be able to drop, delay, or route the traffic as they see fit. All of this assumes that the client handles the certificates properly. Indeed, it has been shown that on smartphones many apps do not handle certificates properly and, as a result, a number of apps are still vulnerable to MITM attacks even when the connection is 'secured' with TLS [38][44].

Remote. Remote attacks are aimed at the victim device from over the network, with the attacker usually being many hops away. Attackers can target network services running on a smartphone, SMS/MMS handlers in the operating system, client-side web browsers, email client applications, and other apps running on the smartphone. Each of these pieces of software can be exploited in different ways.

- ***Network Services***: By default, many smartphones do not have network services installed, but various third-party apps open ports and actively listen for incoming connections. A simple feature in Android, called the Android Debug Bridge can allow access to the device via TCP [33]. Other apps that listen for connections over TCP include Remote Desktop (RDP), Secure Shell (SSH), Virtual Network Computing (VNC) and other similar protocols. Each of these technologies expose a potential area for exploit as they offer a potential means of entry into the smartphone. Open ports can be enumerated by using port-scanning tools such as Nmap to probe the device. Once the list of listening services is obtained, the attacker can then proceed to fingerprint the actual software that is running behind the port before attempting to exploit it. Network services have not received great attention in the literature, and thus network service attacks on smartphones continues to be an open area of research.

- ***Messaging***: In addition to the traditional weak-points that smartphones have as a result of them being Internet-enabled, there are other weak-points that come about from their connectivity to the cellular network. Three notable examples of such weak-points are the Short Message Service (SMS) [46] system, the Multimedia Messaging Service (MMS) [119][123][80] system, and the Wireless Application Protocol (WAP) [60] system. The WAP Push message feature implements a Service Loading (SL) request. This request may cause a smartphone to request a URL. By carefully crafting a web page, Ravishankar Borgaonkar [60] demonstrated how a smartphone can be directed, via Unstructured Supplementary Service Data (USSD) codes, to do things such as lock the SIM card or perform a factory restore. More recently,

the attacks leveraging vulnerabilities in MMS handling came to prominence with the announcement of the 'Stagefright' exploit [80]. This exploit leverages the fact that some devices automatically process a video received by MMS so that it is ready for viewing when the user opens the message. By sending a carefully crafted MMS message, an attacker could potentially perform arbitrary actions on a victim's device through remote code execution, without any interaction required from the victim.

- ***Browser*** - The web browsers installed on smartphones present a rich and dynamic area of potential vulnerability since, in addition to executing their own application logic, they support a number of technologies and protocols as well, such as Javascript. Javascript itself is an entire scripting language and contributes to the complexity of the code that underlies the web browser. Indeed, an entire spectrum of Javascript attacks exist for exploiting browser weaknesses. The most common way to exploit a browser is the 'drive-by' or 'watering hole' attack whereby an attacker causes the user to visit a malicious URL. This is often done through social engineering. The URL is usually laced with malicious payloads that attack vulnerabilities in the browser. Since a browser is a necessarily complex piece of software handling many technologies, it is a very susceptible piece of the stack that has been exploited several times in the literature [34][73][68]. Worryingly, smartphone browsers may even be at more risk since they may not be a mature as their desktop counterparts. However, due to the strict nature of app sandboxing, smartphone browser exploitation may have limited return on investment after successful exploitation.

- ***Email clients*** - Email clients are another potential vulnerable area on smartphones. Attacks exploiting vulnerabilities in email clients can be delivered *en masse*, for example, by using unsolicited email as a vector for delivery. Since many email clients typically render email based on the HTML tags and rich multimedia that they contain, attacks on email clients are possible by exploiting the improper handling of malformed messages. Additionally, email clients usually pass on downloaded attachments to other software for handling which make the email client itself a potential vector for malicious email attachments to be passed on to other software running on the system. Attacks on smartphone email clients are not common and may be an interesting area for future research.

- ***Third-party Apps*** - Third-party apps contribute in several ways to allowing attackers to breach security or invade privacy on smartphones. While the apps available in the official app stores are vetted to varying extents, malicious apps sometimes slip past these protection mechanisms [113]. Moreover, users sometimes opt to install apps from third-party marketplaces,

which are known to contain greater amounts of grayware/malware [120]. But malware is not the only issue; legitimate third-party applications can be exploited by an adversary and caused to perform malicious actions. For example, by performing a MITM attack on the connection between an app and its server [38], an adversary can potentially instruct the app to perform tasks that are outside of its design and leak sensitive data from a smartphone. Mitigating attacks that leverage third-party apps may be hard given that many of these third-party apps have a legitimate reason to access the sensitive APIs and data that they do on a smartphone. By exploiting weaknesses in these apps, adversaries can obtain access to all the data that the app has permission to access. Alternately, adversaries can also make Trojan-horse apps that actually perform a legitimate task but also pilfer data behind the scenes. These types of apps are not easily identified since they may have legitimate reasons to access the APIs that they do and there is no straightforward way to differentiate them from non-malicious apps. This is an open research problem.

Less-privileged smartphone apps can leverage confused deputy attacks to obtain sensitive data and send it from a device. In a confused deputy attack [67], a rogue app without a particular permission tricks one or more apps with the required permissions into carrying out the task that it is not permitted to do [64]. For example, an Android app that did not have permission to connect to the internet[2] could use an 'Intent' to make the browser app make the connection for it. Colluding apps can also combine each of their sets of granted permissions to achieve a greater overall goal [12]. For example, one app with access to the address book may collude with another app with access to the internet, to achieve the overall goal of surreptitiously sending a user's address book over the internet. Colluding apps can communicate directly or by using covert channels, further adding to the complexity of detecting them. Identifying and taming one or more colluding apps is an interesting area of research for intrusion detection and prevention.

1.5 Smartphone App Marketplaces and Malware

Adversaries do not necessarily need to perform elaborate exploits if their only aim is to pilfer sensitive data from a device. This can be done in a straightforward way with the 'consent' of the user, if the adversary creates a trojan horse application and entices the user to install it. Many apps have legitimate reason to access data and sensitive APIs on a device to provide their functionality. For example, a navigation app has reason to retrieve a user's geographical location and also requires access to the internet to load maps. If this navigation app had a secondary purpose of tracking users, it would be very difficult, if at all possible, to identify this sort of malfeasance.

[2]This example is an over-simplification. We note that in iOS/Android 6, permission to use the internet is granted to apps by default.

To this end, malware authors are developing and publishing grayware apps with dubious behaviour to further their malevolent goals.

This worrying trend has not gone unnoticed and indeed the literature [39, 78, 10, 17] is replete with examples of malicious apps making it into both the Google Play Store and Apple App Store. Much harder to detect and remove are grayware apps like Trojan horse apps that are not outrightly malicious but instead mask their malfeasance under the guise of providing some reasonable functionality [122][16]. Malicious (or Trojan horse apps) are able to perform a variety of actions on a smartphone and are only limited by sandboxing and the permissions that they are allowed to perform. However, some malicious apps also contain operating system exploits which they can deploy to break out of their sandbox and/or perform privilege escalation [55]. Many authors have proposed solutions to identify malware in app marketplaces. For example, Chakradeo et al. [15] used statistical methods to measure correlations in app characteristics which reduced the time taken to scan a marketplace.

Repackaged apps are variants of legitimate apps that are reverse-engineered and modified to add or change app behaviour. A common tactic is to edit variables in advertisement code or programmatically click ads to route advertising revenue fraudulently to the attacker [21][120]. Additionally, adversaries sometimes rebrand apps (by modifying the app name and icon) and pass them off as their own. This worrying trend underscores the fact that app marketplaces themselves are a large attack vector that are difficult to police. Moreover, users tend to misplace trust in apps coming from app marketplaces [90][11], both official and unofficial, and unwittingly expose themselves to greater risk than on their desktop computers. This problem is compounded by the fact that anti-malware solutions are less widely deployed on smartphones [11]. Thus adversaries have significant motivation to deliver attacks via app marketplaces.

1.6 Attack Vector Mitigation Using IDS/IPS

We now summarise the mitigation strategies that are used to counter attacks against smartphones. This analysis is summarised in Table 1.5. As discussed in Section 1.3, drive-by (or watering hole) attacks are a popular attack vector given that these attacks can (usually) be deployed remotely and target a moderate number of users without much additional effort required. Drive-by attacks typically target vulnerabilities in the operating system or other software running on smartphones. Unsurprisingly then, most drive-by attacks can be thwarted by simply keeping the smartphone operating system and other third-party software up-to-date. However, this is more easily said than done since many smartphones, especially older Android devices, are still widely used, but no longer receive updates from their vendor. Any useful IDS will need to be able to protect older unpatched devices. We elaborate on the phenomena of unpatched devices in Section 1.8.

As discussed in Section 1.5, third party apps offer a low-investment avenue for attackers to get their code running on users' devices. The potentially exploitable user-

Table 1.5: Table showing common attack vectors and their mitigation strategies.

Attack Vector	Mitigation Strategy
Drive-by attack	Software updates to patch existing vulnerabilities.
App ecosystems	Whitelist of apps that can be installed or more intensive app vetting procedures.
Physical attacks	Encrypt all data on a smartphone and use tamper-proof hardware.
Social engineering	User education and software updates (to limit privilege escalation in case of successful attack).

base can be quite significant considering that publishers have turned to buying app reviews [115], to make their app seem more legitimate in app marketplaces. Attackers can opt to use the permissions their apps have been granted to pilfer as much data from the device as possible, or, with some more investment, can embed exploits into their app to elevate the app's privileges while using techniques [52] to ensure that their app is not removed from the app marketplace. Successfully deploying such an app can lead the attacker to obtaining anything from sensitive data on the device to total compromise of the device. The level of effort required varies, depending on whether the attacker just wants to get some personal information or fully compromise devices. These attacks are hard to target specific users, though most times attackers do not have specific targets in mind when leveraging third-party apps to distribute their malfeasance. The danger from third-party apps can be mitigated, in the first instance, by more intensive app vetting in official marketplaces. IDS/IPSs employed by app stores need to be able to detect dynamic code loading, logic bombs, and the like, and they need to be able to do it at scale while not being detectable by the malware itself.

An adversary having physical access through a victim misplacing their device or having it stolen opens up significant avenues of attack for that device. The low investment required to access misplaced devices makes theft attractive to unskilled adversaries. However, skilled and resourceful adversaries may also target specific devices, such as company-issued devices, in an effort to obtain valuable intellectual property or inside information. If a device is lost/stolen, an adversary has immediate access to all unencrypted data on that device and may access it by directly connecting to memory cards. In the case where the device was not sufficiently secured using screen-lock codes, the adversary would have access to all the other features of the device and could impersonate the owner of the device. Researchers found, in one study, that 29% of participants failed to use a screen-lock on their devices and, in general, underestimate how much personal information is stored on their device [36]. Many physical attacks can be easily mitigated by user training, use of full device encryption, and use of screen-lock functions. However, other attacks that target the underlying circuitry to circumvent protection mechanisms must be remedied using alternative strategies such as tamper-proof hardware.

Finally, social engineering can be used to gain entry into a smartphone. Since

Table 1.6: Mitigation features used to increase the level of investment required by an attacker to successfully exploit a smartphone.

Mitigation Feature	Weak Point Defended
Sandboxing	Browser, operating system, third-party apps
DEP	Browser, operating system, third-party apps
ASLR	Browser, operating system, third-party apps
Verified boot	Operating system, firmware
Cryptography	SIM card, memory card, firmware, operating system

modern smartphone operating systems have several robust security features, attackers have turned to manipulating the user of a device to further their goals. By tricking a user into installing third-party apps or leading them to spoofed web pages, an attacker can obtain access to sensitive device APIs (if a user is tricked into granting an app sensitive permissions) or other privileged information. Social engineering requires varying effort depending on the nature of the attack and how elaborate it is, with the potential gain often being proportional to the social engineering effort required. These non-technical attacks are mitigated by user education (about social engineering and how to remain safe) and software updates to mitigate privilege escalation if the attacker successfully gained an entry point into the smartphone. Intrusion detection and prevention tackling social engineering attacks on smartphones is an open research area.

1.7 Inherited Weak Points and Countermeasures

A critical part of any analysis of smartphone attacks and attack vectors has to do with understanding the similarities and differences in attacking smartphones as opposed to traditional desktops/workstations. In some ways, attacks are of a similar form and are merely adapted to work on smartphones. In other ways, idiosyncrasies of smartphones make attacks on them easier to happen and harder to defend against. For example, in contrast to the typical desktop, the ability to monitor and disrupt attacks on smartphones is reduced since they have multiple points for traffic ingress and egress. Additionally, high mobility (and consequent ease of misplacement) and the plethora of sensors that make smartphones unique also contribute to the rich attack surface they contain.

On a workstation, it usually suffices to install security software such as a firewall and anti-malware solution. Smartphones have many more ways of connecting to the outside world using various wireless technologies and ports and thus a typical firewall approach is no longer sufficient. Trivially, smartphones are more easily lost or misplaced and this offers unique attack vectors to an adversary who stole or otherwise happened upon a device. Full device encryption and screen-locks reduce the potential profit to an adversary that steals or otherwise obtains physical access to a device.

1.7.1 Built-in Mitigation Strategies

Modern smartphone platforms often contain advanced features that mitigate the likelihood of a successful compromise by an adversary. Indeed, smartphones utilise features such as sandboxing, Data Execution Prevention (DEP), Address Space Layout Randomisation (ASLR), and verified boot that add to the complexity of the task of an adversary intent on exploiting a smartphone. Table 1.6 summarises the most common attack mitigation technologies used on smartphones as well as the attack surface that they defend. These technologies are all inherited from modern desktop/server operating systems.

Sandboxing is a well-known security mechanism that is also used on smartphone operating systems for separating running programs. Apps running in a sandbox may only access a tightly controlled set of resources as arbitrated by the operating system. Any additional resources required are accessed through well-defined APIs and, in many cases, apps need to have their intentions to access 'third-party' resources declared, *a priori*, to the operating system. DEP is another feature borrowed from modern computer operating systems for use on smartphones. DEP demarcates areas of memory as containing data that is executable or non-executable. This protects against malicious exploits such as buffer overflow attacks that store executable instructions in a data area of memory. ASLR is typically combined with DEP for even greater security. ASLR randomizes the addresses for key memory areas such as the base of the executable file as well as the stack, heap, and relevant libraries. This makes it very difficult for an attacker to correctly jump to an exploited function in memory and protects against buffer overflow attacks. Verified boot is a hardware and/or software technique concerned with restricting the software that can run on the device during boot up. Verified boot typically only allows software cryptographically signed by the manufacturer to run on the device. This provides an additional layer of security since it detects and prevents potentially compromised software from running on critical parts of the system. In iOS, a secure boot chain ensures that low level software has not been tampered with and that the iOS will only run on validated Apple devices [4].

1.7.2 Attack Vectors and Attack Surfaces on Workstations

A breadth of knowledge already exists with regard to securing the attack surfaces of desktops/workstations. By understanding the similarities and differences with smartphones and workstations, we have an effective foundation from which to understand how best to engineer approaches to secure the attack surfaces on smartphones. Smartphones, in general, contain all the attack surfaces that typical desktops and workstations contain. The smartphone also contains additional attack surfaces which come from the fact that it is also a mobile phone that connects to a cellular network. As a result, smartphones have additional attack surfaces coming from their messaging capability (SMS/MMS which are delivered over the cellular network), their cellular interface (the physical hardware and firmware that is responsible for providing connectivity to cellular networks), and other artefacts of mobile connectivity such as

various ad-hoc communication technologies such as NFC, Bluetooth, and the like. These additional attack surfaces, originating from the idiosyncrasies of smartphone technology, are interesting areas of further research since they do not exist on desktops/workstations and, as such, their protective technologies may not be as mature as those of other attack mitigation systems.

1.8 Related Work and Open Research Problems

By design, smartphones are portable, high connectivity devices with a variety of sensors, technologies, ports, and interfaces. On desktop computers, a firewall can effectively mitigate many attacks since most non-physical attacks come from services exposed to the network. Conversely, on a smartphone, simply installing a firewall can reduce attacks coming from over the network, but those attacks only target a fraction of the vulnerable points present. There are still many non-Internet technologies that run on smartphones, such as NFC, Bluetooth, SMS/MMS, and these continue to provide vulnerable points which cannot be readily 'firewalled'. Engineering IDS solutions for protecting these weak points is an open research challenge.

As we saw in Table 1.4, drive-by attacks are a popular vector of delivering a malicious payload to a smartphone. From Table 1.5 we also saw that the most common mitigation strategy for drive-by attacks is software updates. This is not surprising because most drive-by attacks aim to exploit some software vulnerability on the smartphone. However, getting software patches to end-users is easier said than done. As Thomas et al. [109] discovered, some vulnerabilities will not have been deployed to 95% of vulnerable devices until more than five years after the release of the fix. This leaves a very large window of opportunity for attackers to continue to exploit unpatched devices and this is often no fault of the end-user. Indeed, the Android landscape is highly fragmented [58] meaning that there is no unified way of pushing fixes to Android smartphones automatically. On the other hand, vulnerabilities on Apple and Blackberry devices can be patched more easily since these vendors have greater control over the operating system and update channels. Thus, one of Android's greatest advantages, its open-source nature, is also one of its biggest disadvantages from a security standpoint, in that most Android devices cannot get a security update as soon as it is available. This problem is also made worse on both Android/Windows Phones because of OEMs that modify the operating systems to add their own features, hampering the upgrade process, and sometimes introducing new vulnerabilities themselves [114]. Thus, any IDS should be able to work even on devices with outdated operating systems.

1.8.1 Related Work

Shabtai et al. [6] propose a methodology for evaluating the effectiveness of security solutions on Android. The authors propose evaluation criteria such as visibility, security solution administration, inherent cost, security level and other miscellaneous

artefacts. Given the many security solutions that have been proposed, this work provides an important mechanism for comparing the utility of one solution to another. Along similar lines, Louk et al. [66] argue for the use of monitoring, detecting, tracking and notification (MDTN) as a means of securing against intrusions in smartphone environments. The authors demonstrate the utility of this approach in identifying malware and show that it outperforms some existing approaches.

Shabtai et al. [97] describe a method for intrusion detection on mobile devices using the knowledge-based temporal abstraction (KBTA) methodology. This approach is suitable for identifying previously unknown malware on mobile devices. Their system works by polling the device for particular metrics such as number of SMSs sent, along with critical system events. This data is combined with a knowledge-base that allows an abstraction into high-level patterns. The authors deploy their proposed system as a host-based intrusion detection system (HIDS) and show that it yields a performance rate above 94%. The authors further argue for the utility of their system on battery-constrained devices like smartphones, by showing an average CPU consumption of only 3%.

Houmansadr et al. [53] was one of the first to leverage the cloud for intrusion detection. Zounouz et al. [124] later followed a similar approach and proposed Secloud, a cloud-based security solution for smartphones. Secloud works by emulating a smartphone using cloud resources and sending device input from the device to be protected to this virtual device. In this way, Secloud can perform resource-intensive security analysis without burdening the actual physical device. Secloud's emulator performs virus-scanning, file-integrity checking, system-call monitoring and intrusion detection and response. The authors validate the utility of Secloud by showing that it accurately detects intrusions while consuming negligible resources.

Along similar lines, Shabtai et al. [98] present Andromaly, a malware detection framework for Android. Andromaly is host-based and feeds continuously collected features and events from the target device to anomaly detectors. The anomaly detectors are built around machine learning classifiers. The authors evaluate several classification algorithms and feature selection methods. They show that malware can be detected in a way that is both lightweight and accurate. Shabtai et al. [99], in related work, propose a behaviour-based anomaly detection system that identifies anomalies based on traffic patterns.

More recently, Ariyapala et al. [5] combine host and network metrics to build an intrusion detection system for smartphones. The authors capture metrics such as CPU utilisation, energy consumption, running processes, user activity and network traffic. Damopoulos et al. [28] propose a framework that unifies host- and cloud-based intrusion detection systems. The authors validate their system by showing that it can deliver quick and accurate results using computations that are affordable on an iPhone. Papamartzivanos et al. [88] crowdsource information on privacy leaks from smartphones using a cloud-based architecture. Finally, Damopoulos et al. [29] propose one of the first anomaly-based intrusion detection systems for mobile devices and use iPhone user data to show that their system can detect intrusions with up to a 99.8% true positive rate.

1.8.2 Open Research Problems

Smartphones are devices designed with seamless connectivity in mind. Thus these devices are shipped with a wide variety of wireless interfaces powered by various technologies. Since smartphones are utilised by users of varying technical skill, manufacturers may be tempted to sacrifice security to appease the consumer. Although there is the 'walled-garden' approach being taken with the app repositories, apps of dubious intent, but seemingly legitimate, are an increasing problem. Indeed, third-party apps that access sensitive data on a device are free to package it and send it over the Internet. The destination or reason for doing this may not be immediately clear after static or dynamic analysis so there is still a risk of data theft by third-party apps that were entrusted to carry out a specific task. This problem is further complicated by confused deputy attacks and apps that collude to avoid detection. IDS solutions that identify and mitigate covert channels for app collusion remains a challenge.

From our survey of the literature, we uncovered that email client apps and network service apps (those that open ports) have not received wide attention from adversaries or the research community. These categories of apps may potentially be more vulnerable because of their distinctive characteristics. For email apps, the rending of HTML email or automatic downloading of attachments can provide unique access to the system if malicious emails/attachments are not handled properly. Network service apps that open ports on a smartphone may also introduce vulnerabilities if they are not designed properly. Even very mature network services on desktops/workstations/servers contain vulnerabilities, so it would be no surprise if the less mature smartphone versions of these services also contain vulnerabilities. This fact becomes worrying when one considers that many apps are developed by small teams or individuals with potentially little knowledge or concern for security. Sub-module firmware (such as Wi-Fi) has also received little attention from attackers. This may be because it requires esoteric knowledge to actually interface with hardware. If successfully exploited, sub-module firmware can provide long-term, almost undetectable, elevated privileges on a smartphone. IDS approaches to protecting hardware are a welcome area for future research.

More recent additions to smartphones such as NFC communication and the use of biometrics for authentication introduce new areas of potential vulnerability to be exploited. Given that these features are increasingly being used for making purchases using a smartphone, it seems natural that the attention of adversaries would shift in this direction. Indeed, the literature shows researchers who were capable of intercepting data from contactless payment cards [31]. Biometric authentication on mobile devices is also an area worth exploring since keeping credentials safe while allowing secure authentication on devices that are easily lost is an ongoing challenge.

Rogue cellular base stations are also a challenge due to the availability and rapidly-falling prices of cellular infrastructure such as picocells. Combined with the prevalence of unpatched devices with gaping security holes, rogue base stations have the potential to quickly and silently compromise many devices at once. Out-of-date devices also increase the likelihood of smartphone worms that propagate by exploiting wireless interfaces for transmission. As mentioned earlier in this chapter, IDS

systems will need to continue to offer protection even on devices that are no longer supported by their manufacturer.

1.9 Conclusion

Smartphones are poised to take over from desktops and workstations as the device of choice for communication, shopping, banking and web browsing. As these devices become pervasive, the interest of adversaries naturally turns in that direction, as the adversaries look at ways of exploiting users and stealing data to make a profit. In this chapter, we enumerated the various assets on a smartphone, as well as the attack surfaces that are present on these devices that can be used by an adversary to gain entry into the system. Until now, it was not well understood by a non-expert how the various components of a smartphone collectively contributed to its overall attack surface. We showed that smartphones have all the attack surfaces that desktops do, with additional attack surfaces coming from the fact that they contain additional hardware for mobility, sensing, and ad-hoc connectivity. We discuss the various attack mitigation features in use on smartphones, some directly borrowed from desktops/workstations, and highlight the various weak-points that are defended using these technologies. We also analysed, in general, how vulnerabilities on smartphones can be reduced, but also demonstrate how some operating systems, like Android, will naturally have more challenges in getting software updates to users. As smartphones become a more tightly-knit part of the average person's life, if is important to have a solid grasp of the ways that these devices are vulnerable, so that we can continue to develop ways of keeping end-users safe, now and into the foreseeable future.

References

[1] Andrey Polkovnichenko. BrainTest A New Level of Sophistication in Mobile Malware. http://blog.checkpoint.com/2015/09/21/braintest-a-new-level-of-sophistication-in-mobile-malware/, September 2015.

[2] Android. Security-Enhanced Linux in Android. https://source.android.com/security/selinux/index.html.

[3] Apple Inc. App Review. https://developer.apple.com/app-store/review/.

[4] Apple Inc. iOS Security - White Paper. https://www.apple.com/business/docs/iOS_Security_Guide.pdf, September 2015.

[5] K. Ariyapala, H. G. Do, H. N. Anh, W. K. Ng, and M. Conti. A Host and Network Based Intrusion Detection for Android Smartphones. In *30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 849–854, March 2016.

[6] Yuval Elovici Asaf Shabtai, Dudu Mimran. Evaluation of Security Solutions for Android Systems. https://arxiv.org/abs/1502.04870v1.

[7] Adam J Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M Smith. Smudge Attacks on Smartphone Touch Screens. *WOOT*, 10:1–7, 2010.

[8] Kiran Bala, Sumit Sharma, and Gurpreet Kaur. A Study on Smartphone based Operating System. *International Journal of Computer Applications*, 121(1), 2015.

[9] Jethro G Beekman and Christopher Thompson. Man-in-the-middle attack on T-Mobile Wi-Fi Calling. Technical report, University of California at Berkeley, 2013.

[10] Karissa Bell. Everything you need to know about App Store malware. http://mashable.com/2015/09/21/ios-app-store-malware/, September 2015.

[11] Zinaida Benenson and Lena Reinfelder. Should the Users be Informed? On Differences in Risk Perception between Android and iPhone Users. In *Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[12] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. Towards Taming Privilege-Escalation Attacks on Android. In *NDSS*, 2012.

[13] Liang Cai and Hao Chen. TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion. In *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, HotSec'11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association.

[14] MJ Callaghan, Jim Harkin, TM McGinnity, et al. Case study on the Bluetooth vulnerabilities in mobile devices. *IJCSNS International Journal of Computer Science and Network Security*, 6(4):125–129, 2006.

[15] Saurabh Chakradeo, Bradley Reaves, Patrick Traynor, and William Enck. MAST: Triage for Market-scale Mobile Malware Analysis. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 13–24, New York, NY, USA, 2013. ACM.

[16] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is This App Safe?: A Large Scale Study on Application Permissions and Risk Signals. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 311–320, New York, NY, USA, 2012. ACM.

[17] Graham Cluley. Malware hits the Google Play Android app store again (and again). https://grahamcluley.com/2015/09/video-malware-hits-google-play-android-app-store/, September 2015.

[18] Patrick Collinson. Don't bank on your phone,it could be hacked by Zeus 'trojan horse' . http://www.theguardian.com/money/2011/jul/22/smartphones-hacked-zeus-malware, July 2011.

[19] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing Android Encrypted Network Traffic to Identify User Actions. *IEEE Transactions on Information Forensics and Security*, 11(1):114–125, Jan 2016.

[20] Mauro Conti, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Can't You Hear Me Knocking: Identification of User Actions on Android Apps via Traffic Analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, pages 297–304, New York, NY, USA, 2015. ACM.

[21] Jonathan Crussell, Ryan Stevens, and Hao Chen. MAdFraud: Investigating Ad Fraud in Android Applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 123–134, New York, NY, USA, 2014. ACM.

[22] CVE Details. Apple Iphone OS: List of security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-49/product_id-15556/Apple-Iphone-Os.html.

[23] CVE Details. Blackberry OS: List of security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-8356/product_id-25529/Blackberry-Blackberry-Os.html.

[24] CVE Details. Google Android: List of security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html.

[25] CVE Details. Microsoft Windows Mobile: List of security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-9709/Microsoft-Windows-Mobile.html.

[26] Rajendra M Dabhi and Sunil Kumar V Nakum. A Paper on Latest and Upcoming Smartphone OS. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(4), 2014.

[27] Shuaifu Dai, A. Tongaonkar, Xiaoyin Wang, A. Nucci, and D. Song. NetworkProfiler: Towards automatic fingerprinting of Android apps. In *INFOCOM, 2013 Proceedings IEEE*, pages 809–817, April 2013.

[28] Dimitrios Damopoulos, Georgios Kambourakis, and Georgios Portokalidis. The Best of Both Worlds: A Framework for the Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones. In *Proceedings of the 7th European Workshop on System Security*, EuroSec '14, pages 6:1–6:6, New York, NY, USA, 2014. ACM.

[29] Dimitrios Damopoulos, Sofia A Menesidou, Georgios Kambourakis, Maria Papadaki, Nathan Clarke, and Stefanos Gritzalis. Evaluation of Anomaly-based IDS for Mobile Devices Using Machine Learning Classifiers. *Security and Communication Networks*, 5(1):3–14, 2012.

[30] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable. In *NDSS*, 2014.

[31] Thomas P. Diakos, Johann A. Briffa, Tim W. C. Brown, and Stephan Wesemeyer. Eavesdropping near-field contactless payments: a quantitative analysis. *The Journal of Engineering*, January 2013.

[32] Erich Dondyk, Louis Rivera, and Cliff C Zou. Wi–Fi access denial of service attack to smartphones. *International Journal of Security and Networks*, 8(3):117–129, 2013.

[33] Joshua J Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A Ridley, and Georg Wicherski. *Android Hacker's Handbook*. John Wiley & Sons, 2014.

[34] Thomas Dullien. Ralf-Philipp Weinmann & Vincenzo Iozzo own the iPhone at PWN2OWN. http://blog.zynamics.com/2010/03/24/ralf-philipp-weinmann-vincenzo-iozzo-own-the-iphone-at-pwn2own/, March 2010.

[35] Graham Edgecombe. Detection of SSL-related security vulnerabilities in Android applications. https://grahamedgecombe.com/talks/android-ssl.pdf, 2014.

[36] Serge Egelman, Sakshi Jain, Rebecca S. Portnoff, Kerwell Liao, Sunny Consolvo, and David Wagner. Are You Ready to Lock? In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 750–761, New York, NY, USA, 2014. ACM.

[37] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A Study of Android Application Security. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[38] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 50–61, New York, NY, USA, 2012. ACM.

[39] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in Smartphones and Mobile Devices*, pages 3–14. ACM, 2011.

[40] Fraunhofer Institute for Secure Information Technology. Smartphone Attack Vectors. https://www.sit.fraunhofer.de/fileadmin/dokumente/poster/Smartphone-Attack-Vectors_Poster_Fraunhofer-SIT.pdf.

[41] Gartner. Gartner Says Emerging Markets Drove Worldwide Smartphone Sales to 15.5 Percent Growth in Third Quarter of 2015. http://www.gartner.com/newsroom/id/3169417, November 2015.

[42] Gartner. Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015. http://www.gartner.com/newsroom/id/3215217, February 2016.

[43] Georgia Institute of Technology. Georgia Tech Uncovers iOS Security Weaknesses. http://www.news.gatech.edu/2013/07/31/georgia-tech-uncovers-ios-security-weaknesses, July 2013.

[44] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software. In *Proceedings of the 2012 ACM*

Conference on Computer and Communications Security, CCS '12, pages 38–49, New York, NY, USA, 2012. ACM.

[45] Clint Gibler, Ryan Stevens, Jonathan Crussell, Hao Chen, Hui Zang, and Heesook Choi. AdRob: Examining the Landscape and Impact of Android Application Plagiarism. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 431–444, New York, NY, USA, 2013. ACM.

[46] Nico Golde and Collin Mulliner. Countering SMS Attacks: Filter Recommendations. Technical report, Technische Universität Berlin, 2011.

[47] Gabriel Goller and Georg Sigl. Side Channel Attacks on Smartphones and Embedded Devices Using Standard Radio Equipment. In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design*, volume 9064 of *Lecture Notes in Computer Science*, pages 255–270. Springer International Publishing, 2015.

[48] Guang Gong. Exploiting Heap corruption due to Integer Overflow in Android libcutils. *Black Hat USA*, 2015.

[49] Joan Goodchild. Social engineering: 3 mobile malware techniques. http://www.csoonline.com/article/2129129/social-engineering/social-engineering–3-mobile-malware-techniques.html, July 2011.

[50] Jin Han, SuMon Kywe, Qiang Yan, Feng Bao, Robert Deng, Debin Gao, Yingjiu Li, and Jianying Zhou. Launching Generic Attacks on iOS with Approved Third-Party Applications. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 272–289. Springer Berlin Heidelberg, 2013.

[51] Andrew Hoog. *Android forensics: investigation, analysis and mobile security for Google Android*. Elsevier, 2011.

[52] Oliva Hou. A Look at Google Bouncer. http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/, July 2012.

[53] Amir Houmansadr, Saman A. Zonouz, and Robin Berthier. A Cloud-based Intrusion Detection and Response System for Mobile Phones. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, DSNW '11, pages 31–32, Washington, DC, USA, 2011. IEEE Computer Society.

[54] John Hubbard, Ken Weimer, and Yu Chen. A study of SSL Proxy attacks on Android and iOS mobile applications. In *11th IEEE Consumer Communications and Networking Conference (CCNC) 2014*, pages 86–91. IEEE, 2014.

[55] Xuxian Jiang. GingerMaster: First Android Malware Utilizing a Root Exploit on Android 2.3 (Gingerbread). http://www.csc.ncsu.edu/faculty/jiang/GingerMaster/, 2011.

[56] P. Kaur and S. Sharma. Google android a mobile platform: A review. In *Recent Advances in Engineering and Computational Sciences (RAECS) 2014*, pages 1–5, March 2014.

[57] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology - CRYPTO' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer Berlin Heidelberg, 1999.

[58] Steve Kovach. Android is facing a security crisis. http://uk.businessinsider.com/stagefright-vulnerability-is-bad-news-for-android-2015-8, August 2015.

[59] Krebson Security. Beware of Juice-Jacking. http://krebsonsecurity.com/2011/08/beware-of-juice-jacking/, August 2011.

[60] Richard Lardner. Ravi Borgaonkar Says Android Phones Vulnerable To Wipeout Attack. http://www.huffingtonpost.com/2012/09/30/ravi-borgaonkar-says-andr_n_1923867.html, Sep 2012.

[61] Billy Lau, Yeongjin Jang, Chengyu Song, Tielei Wang, and PH Chung. Mactans: Injecting Malware into iOS Devices via Malicious Chargers. In *Black Hat USA*, 2013.

[62] Gary Legg. The Bluejacking, Bluesnarfing, Bluebugging Blues: Bluetooth Faces Perception of Vulnerability. http://www.eetimes.com/document.asp?doc_id=1275730, 2005.

[63] Robert LiKamWa, Yunxin Liu, Nicholas D. Lane, and Lin Zhong. MoodScope: Building a Mood Sensor from Smartphone Usage Patterns. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 389–402, New York, NY, USA, 2013. ACM.

[64] Anthony Lineberry, David Luke Richardson, and Tim Wyatt. These arent the permissions youre looking for. *DefCon*, 18:2010, 2010.

[65] Hiroshi Lockheimer. Android and Security. http://googlemobile.blogspot.co.uk/2012/02/android-and-security.html, February 2012.

[66] Maya Louk, Hyotaek Lim, and HoonJae Lee. An Analysis of Security System for Intrusion in Smartphone Environment. *The Scientific World Journal*, 2014, 2014.

[67] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 229–240, New York, NY, USA, 2012. ACM.

[68] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. Attacks on WebView in the Android system. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 343–352. ACM, 2011.

[69] Weizhi Meng, Wang Hao Lee, S.R. Murali, and S.P.T. Krishnan. Charging Me and I Know Your Secrets!: Towards Juice Filming Attacks on Smartphones. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, CPSS '15, pages 89–98, New York, NY, USA, 2015. ACM.

[70] Mike Brown. Windows 10 'Redstone': 64-bit ARM Support Could Deliver Fastest Windows Phones Ever. http://www.ibtimes.com/windows-10-redstone-64-bit-arm-support-could-deliver-fastest-windows-phones-ever-2267877, January 2016.

[71] Charlie Miller. Exploring the NFC attack surface. *Black Hat USA 2012*, 2012.

[72] Charlie Miller, Dion Blazakis, Dino DaiZovi, Stefan Esser, Vincenzo Iozzo, and Ralf-Philip Weinmann. *iOS Hacker's Handbook*. John Wiley & Sons, 2012.

[73] Charlie Miller and Vincenzo Iozzo. Fun and games with Mac OS X and iPhone payloads. *Black Hat Europe*, 2009.

[74] Ken Munro. Android scraping: accessing personal data on mobile devices. *Network Security*, 2014(11):5 – 9, 2014.

[75] MWR InfoSecurity. MWR Labs expose vulnerabilities in Samsung Galaxy S5 and Amazon Fire Phone. https://www.mwrinfosecurity.com/media/press-releases/mwr-labs-expose-vulnerabilities-in-samsung-galaxy-s5-and-amazon-fire-phone/, 2014.

[76] MWR InfoSecurity. Sandbox bypass through Google Admin WebView. https://labs.mwrinfosecurity.com/system/assets/1021/original/mwri-advisory_sandbox_bypass_through_google_admin_webview.pdf, August 2015.

[77] MWR Labs. Android 4.4.2 Secure USB Debugging Bypass. https://labs.mwrinfosecurity.com/advisories/android-4-4-2-secure-usb-debugging-bypass/.

[78] Alexios Mylonas, Stelios Dritsas, Bill Tsoumas, and Dimitris Gritzalis. Smartphone security evaluation The malware attack case. In *Proceedings of the International Conference on Security and Cryptography (SECRYPT) 2011*, pages 25–36. IEEE, 2011.

[79] National Security Agency. New Smartphones and the Risk Picture. https://www.nsa.gov/ia/_files/factsheets/mobilerisks.pdf, April 2012.

[80] Phil Nickinson. The 'Stagefright' exploit: What you need to know. http://www.androidcentral.com/stagefright, Aug 2015.

[81] Nielson. Smartphones: So Many Apps, So Much Time. http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps–so-much-time.html, July 2014.

[82] Masoud Nosrati, Ronak Karimi, and Hojat Allah Hasanvand. Mobile computing: principles, devices and operating systems. *World Applied Programming*, 2(7):399–408, 2012.

[83] John Oberheide. When angry birds attack: Android edition. https://www.duosecurity.com/blog/when-angry-birds-attack-android-edition, May 2011.

[84] John Oberheide. Dissecting the Android Bouncer. https://jon.oberheide.org/blog/2012/06/21/dissecting-the-android-bouncer/, June 2012.

[85] Jon Oberheide. Android hax. *Proceedings of SummerCon 2010*, June 2010.

[86] OO Okediran, OT Arulogun, RA Ganiyu, and CA Oyeleye. Mobile Operating Systems and Application Development Platforms: A Survey. *International Journal of Advanced Networking and Applications*, 6(1):2195–2201, 2014.

[87] OpenBTS. OpenBTS - Open source Cellular Infrastructure. http://openbts.org/.

[88] Dimitrios Papamartzivanos, Dimitrios Damopoulos, and Georgios Kambourakis. A Cloud-based Architecture to Crowdsource Mobile App Privacy Leaks. In *Proceedings of the 18th Panhellenic Conference on Informatics*, PCI '14, pages 59:1–59:6, New York, NY, USA, 2014. ACM.

[89] Min-Woo Park, Young-Hyun Choi, Jung-Ho Eom, and Tai-Myoung Chung. Dangerous Wi-Fi Access Point: Attacks to Benign Smartphone Applications. *Personal Ubiquitous Computing*, 18(6):1373–1386, August 2014.

[90] Dhanya Pramod and Ramakrishnan Raman. A Study on the User Perception and Awareness of Smartphone Security. *International Journal of Applied Engineering Research, ISSN*, pages 0973–4562, 2014.

[91] Rapid7. Android Browser and WebView addJavascriptInterface Code Execution. https://www.rapid7.com/db/modules/exploit/android/browser/webview_addjavascriptinterface.

[92] Rapid7. Apple iOS MobileSafari LibTIFF Buffer Overflow. https://www.rapid7.com/db/modules/exploit/apple_ios/browser/safari_libtiff.

[93] Chris Rose. Ubiquitous smartphones, zero privacy. *Review of Business Information Systems (RBIS)*, 16(4):187–192, 2012.

[94] Roman Schlegel, Kehuan Zhang, Xiao-yong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *NDSS*, volume 11, pages 17–33, 2011.

[95] Patrick Schulz and D Plohmann. Android security-common attack vectors. *Rheinische Friedrich-Wilhelms-Universitat Bönn, Germany, Tech. Rep*, 2012.

[96] Nicolas Seriot. iPhone privacy. *Black Hat DC*, page 30, 2010.

[97] Asaf Shabtai, Uri Kanonov, and Yuval Elovici. Intrusion Detection for Mobile Devices Using the Knowledge-based, Temporal Abstraction Method. *J. Syst. Softw.*, 83(8):1524–1537, August 2010.

[98] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "Andromaly": A Behavioral Malware Detection Framework for Android Devices. *J. Intell. Inf. Syst.*, 38(1):161–190, February 2012.

[99] Asaf Shabtai, Lena Tenenboim-Chekina, Dudu Mimran, Lior Rokach, Bracha Shapira, and Yuval Elovici. Mobile Malware Detection Through Analysis of Deviations in Application Network Behavior. *Computers & Security*, 43:1–18, 2014.

[100] TN Sharma, Mahender Kr Beniwal, and Arpita Sharma. Comparative Study of Different Mobile Operating Systems. *International Journal of Advancements in Research & Technology*, 2:1–2, 2013.

[101] Raul Siles. Real world ARP spoofing. Technical report, SANS Institute, 2003.

[102] Namheun Son, Yunho Lee, Dohyun Kim, Joshua I James, Sangjin Lee, and Kyungho Lee. A study of user data integrity during acquisition of Android devices. *Digital Investigation*, 10:S3–S11, 2013.

[103] Raphael Spreitzer. Pin Skimming: Exploiting the ambient-light sensor in mobile devices. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 51–62. ACM, 2014.

[104] Statista. Number of available applications in the Google Play Store from December 2009 to July 2015. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, 2015.

[105] Statista. Number of available apps in the Apple App Store from July 2008 to June 2015. http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/, 2015.

[106] Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. Who Do You Sync You Are?: Smartphone Fingerprinting via Application Behaviour. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless*

and Mobile Networks, WiSec '13, pages 7–12, New York, NY, USA, 2013. ACM.

[107] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. AppScanner: Automatic Fingerprinting of Smartphone Apps From Encrypted Network Traffic. In *1st IEEE European Symposium on Security and Privacy (Euro S&P)*, 2016.

[108] Jennifer Thom-Santelli, Alex Ainslie, and Geri Gay. Location, location, location: A study of bluejacking practices. In *CHI '07 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '07, pages 2693–2698, New York, NY, USA, 2007. ACM.

[109] Daniel R Thomas, Alastair R Beresford, Thomas Coudray, Tom Sutcliffe, and Adrian Taylor. The Lifetime of Android API vulnerabilities: case study on the JavaScript-to-Java interface.

[110] Daniel R. Thomas, Alastair R. Beresford, Thomas Coudray, Tom Sutcliffe, and Adrian Taylor. *23rd International Workshop on Security Protocols*, chapter The Lifetime of Android API Vulnerabilities: Case Study on the JavaScript-to-Java Interface, pages 126–138. Springer International Publishing, Cham, 2015.

[111] Steven J. Vaughan-Nichols. Smartphone operating systems: The rise of Android, the fall of Windows. http://www.zdnet.com/article/smartphone-operating-systems-the-rise-of-android-the-fall-of-windows/, February 2013.

[112] Timothy Vidas, Daniel Votipka, and Nicolas Christin. All Your Droid Are Belong to Us: A Survey of Current Android Attacks. In *WOOT*, pages 81–90, 2011.

[113] Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, and Wenke Lee. Jekyll on iOS: When Benign Apps Become Evil. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 559–572, Berkeley, CA, USA, 2013. USENIX Association.

[114] Lei Wu, Michael Grace, Yajin Zhou, Chiachih Wu, and Xuxian Jiang. The Impact of Vendor Customizations on Android Security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*, CCS '13, pages 623–634, New York, NY, USA, 2013. ACM.

[115] Zhen Xie and Sencun Zhu. Appwatcher: Unveiling the underground market of trading mobile app reviews. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 10:1–10:11, New York, NY, USA, 2015. ACM.

[116] Zhi Xu, Kun Bai, and Sencun Zhu. TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-board Motion Sensors. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 113–124, New York, NY, USA, 2012. ACM.

[117] Yu Yu, Junrong Liu, F-X Standaert, Zheng Guo, Dawu Gu, Sun Wei, Yijie Ge, and Xinjun Xie. Cloning 3G/4G SIM Cards with a PC and an Oscilloscope: Lessons Learned in Physical Security. In *Black Hat USA*, 2015.

[118] Hui Xue Tao Wei Yulong Zhang, Zhaofeng Chen. Fingerprints On Mobile Devices: Abusing and Leaking. https://www.blackhat.com/docs/us-15/materials/us-15-Zhang-Fingerprints-On-Mobile-Devices-Abusing-And-Leaking-wp.pdf.

[119] Z Team. Experts Found a Unicorn in the Heart of Android. https://blog.zimperium.com/experts-found-a-unicorn-in-the-heart-of-android/, July 2015.

[120] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY '12, pages 317–326, New York, NY, USA, 2012. ACM.

[121] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *2012 IEEE Symposium on Security and Privacy (SP)*, pages 95–109, May 2012.

[122] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *NDSS*, 2012.

[123] zLabs. Zimperium zLabs is Raising the Volume: New Vulnerability Processing MP3/MP4 Media. https://blog.zimperium.com/zimperium-zlabs-is-raising-the-volume-new-vulnerability-processing-mp3mp4-media/, October 2015.

[124] Saman Zonouz, Amir Houmansadr, Robin Berthier, Nikita Borisov, and William Sanders. Secloud: A Cloud-based Comprehensive and Lightweight Security Solution for Smartphones. *Comput. Secur.*, 37:215–227, September 2013.