Mobile Security:

Challenges, Lessons, and Future Directions

By Hassen Saïdi and Ashish Gehani

Mobile computing differs fundamentally from its precursors. The computing industry is at an inflection point, with more tablets being shipped than desktops for the first time in 2013. The authors discuss current approaches to mobile computing security, lessons learned about the limitations of these solutions, and promising avenues for future research.

Abstract

Mobile computing differs fundamentally from its precursors. Users entrust their devices with troves of privacy-sensitive data. Devices are utilized for both corporate as well as personal purposes. Computation is limited by battery power rather than processor or communication speeds. Devices can join networks at arbitrary locations and leave at any point in time. Consequently the security challenges are commensurately novel. This is particularly noteworthy since the computing industry is at an inflection point, with more tablets being shipped than desktops for the first time in 2013.

We discuss current approaches to mobile computing security, lessons learned about the limitations of these solutions, and promising avenues for future research.

In 2013, for the first time ever, the total number of smartphones shipped worldwide is projected to surpass the total number of feature phones shipped. It is already the case that mobile devices in general are being shipped in greater numbers than desktops. Over a billion smartphones and tablets are being shipped in 2013 worldwide. It is believed that by 2015, most of the Internet traffic will be driven by mobile devices. They run a total of over two million mobile applications provided by official and unofficial application stores. Mobile devices are being adopted in the enterprise by a larger number of people at a greater pace than ever before. Mobile devices are characterized by a rich set of sensors that provide a variety of important and useful features. They are largely built using legacy operating systems that are modified to accommodate the performance constraints of resource-limited platforms. In addition to commodity operating systems, mobile platforms include rich software frameworks that facilitate access for an ever increasing number of embedded sensors and applications. Because of the blend of code, mobile platforms are subject to failures to secure and harden commodity software as well as failures to understand the security ramification of the ever increasing new applications. Therefore, they represent a constant security challenge for individual users as well as enterprises since "bring your own device" (BYOD) is becoming the rule rather than the exception in today's workplace.

As enterprises and individuals embrace cloud computing, their most trusted data is migrating to systems that are continuously accessible via mobile devices. The security of this data thus depends on the extent to which the devices can be protected. This gives rise to a number of issues. The proliferation of mobile applications makes it difficult for users to track

and understand the danger of their security or privacy being breached by third-party code. Enterprises have decreasing control over the computing platform as the BYOD model gains ground. When these devices have closed platforms, third-party security solutions cannot be developed and enterprises have limited control over the security of the software. On the other hand, the open platforms suffer from fragmentation in the market (and the negative impact on how much testing can be performed for each variant) and slow update cycles (with vulnerabilities remaining unpatched in the wild for longer periods).

With the dwindling role of Blackberry as the leading enterprise phone, most mobile devices run three commodity operating systems: Apple's iOS, Google's Android, and Microsoft's Windows. This year, the Pentagon approved [DoD] the use of iPads, iPhones, and other Apple products as well as Samsung devices running Android, by its soldiers, sailors, and pilots. Android is the leading mobile platform in the US and is driving much of the growth of mobile device usage worldwide. With over fifty percent of market share in the US and with over seventy percent in emerging markets like China, Android is an established platform that has also attracted a lot of malicious activity. With nearly a million mobile applications (apps) developed for Android, malware is on the rise. Known instances of Android-related malware have jumped steadily month by month from 400 in June 2011 to 15,507 in February 2012, according to Juniper Networks [Jun12]. In August 2011, Lookout Mobile Security found that "an estimated half million to one million people were affected by Android malware in the first half of 2011" [Loo11], all from malicious apps. Recently security company Trend Micro claimed that roughly one in ten apps on the Google Play app store was outright malicious [Tre13]. Identifying malicious apps is not an obvious task. For instance, the transmission of private data such as contact lists, or IMEI, ICCID, and telephone numbers may well be for legitimate reasons. Determining for each app whether such an action is done for a legitimate purpose or not is not trivial. iOS has its share of vulnerabilities and malware. The Apple store regularly rejects up to 7% of submitted applications because they violate Apple's mobile application guidelines. Many of the rejected applications are found to invoke privileged application programming interfaces (APIs) that are deemed to escalate the privilege of applications and may violate a user's privacy. The number of malware instances targeting iOS is far smaller than that for Android, as the locked nature of the iOS platform makes it harder to create secondary unregulated markets.

Challenges

There are three significant reasons why mobile platforms represent a security challenge.

New security and privacy concerns

Mobile platforms present us with new challenges because of the combination of a rich application layer and an ever increasing set of sensors. Unlike desktop users, mobile users rely on a larger number of untrusted applications that have access to a large amount of private information.

Recycling technology

Not all previously proposed security mechanisms apply to the mobile world. Firewalls and antivirus systems might have been very popular in the desktop world, but one can argue that they are not only inadequate for the mobile world but also have never been sufficient and effective enough in their original settings.

Mobile platform management

Finally, mobile platforms are not as open and customizable as desktops. The control exercised by device manufacturers and operating system providers makes it difficult to incorporate novel security technologies.

In this paper, we discuss current approaches to securing mobile platforms and survey the various developed and deployed technologies. We describe lessons learned about the limitations of these solutions, and promising avenues for future research. We first describe the architecture of modern mobile platforms and describe the various threats associated with them. We then describe what strategies are deployed to mitigate the threats and secure mobile devices. Finally, we describe lessons learned from each approach and promising future research directions.

Attack surfaces

The leading mobile operating systems share a similar design. Figure 1 on the next page illustrates a typical mobile operating system stack. At the top, user applications consist of executables in various formats. Applications consist of compiled code that originates as source in Java, Objective C, or C. Application code typically invokes framework APIs that provide much of the needed functionality to access the services provided by the sensors and other services from the system and preinstalled apps. The framework APIs provide attractive functionality that programmers use to invoke the various distinctive features of smartphones. In addition to the framework API, a set of system libraries is used. These libraries represent a unified and non-bypassable interface to the underlying kernel. The customized kernel includes a set of drivers that facilitates access to the various sensors. At the bottom, the hardware consists of a processor that is often an ARM core, along with graphical processing units (GPUs), and a set of sensors such as a camera or microphone and Bluetooth and Wi-Fi radios.

Mobile apps on all platforms are sandboxed and do not have access to each other's runtime environment. Sandboxing represents the foundation of mobile platform security. However, mobile apps often exchange data through shared services and storage. Such a clash between the "need to share" versus the "need to protect" philosophies drives much of the security problems that arises in mobile platforms. On the one hand, the need to protect motivates the use of sandboxing that ensures apps are isolated from each other and run in

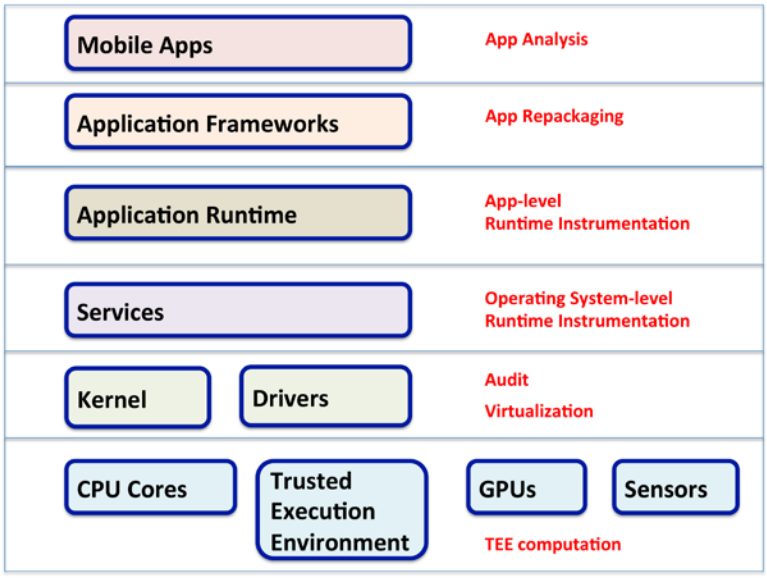

Figure 1: The architecture of a modern mobile platform is depicted here. Defensive technologies are labeled in red and shown in the layer where they are deployed.

separate security domains. On the other hand, the "need to share" aims to minimize app developers' coding effort and promotes the sharing of common capabilities and resources, such as system components like the web browser and storage. In addition to process sandboxing, all access to services and sensors is permission-based, that is, each application must explicitly request access to specific resources.

Root exploits

Perhaps the most dangerous attacks are those that exploit vulnerabilities in the legacy operating systems. Android's Linux kernel, iOS, and the Windows operating system each consists of a large number of lines of code that contain numerous known and unknown vulnerabilities. A malicious app may be able to embed an exploit that allows it to gain root access on the device and control much of the software stack. In particular, it would be able to disable any custom monitoring software that has been installed.

The sandboxing mechanism that ensures the isolation of apps also prevents the antivirus software from actively monitoring an arbitrary app's behavior.

System services

Because the kernel and its services are far removed from the application layer, they are vulnerable to malicious applications that exploit weaknesses in the application permission system. Android is particularly vulnerable to escalations of privilege known as confused-deputy attacks when an app accesses a service for which it does not have an explicit permission. It does this through a request to an application that does have the required permission. All platforms are vulnerable to information leakage through the web browser. An app can easily send information to a remote site even if it does not have access to the network. It does this by just invoking the web browser with an arbitrary URL.

Framework APIs

The rich set of APIs available to mobile apps can be exploited by malicious apps. Without gaining root access or privilege escalation, an app could abuse the services it has access to. For instance, a social network application might have access to GPS data for location check-in purposes. However, a malicious social networking app could continuously collect GPS readings to track the user without consent and even if the user has not invoked the check-in functionality of the app.

Sandboxing

While apps run in sandboxes, their data is often not sandboxed; that is, while applications do not have access to each other's runtime environment or local data, data is often located in shared resources that are not sandboxed. This is a problem for platforms that use shared storage designed for removable content. Such content is produced by different apps and if not encrypted, it can be read by all apps that have access to the shared storage.

Mobile security approaches

Because of the complexity of mobile platforms' software stack, a single approach to securing the platforms and defending against all possible threats has not been advocated. Instead, multiple efforts targeting specific layers have been proposed, as illustrated in Figure 1.

App vetting

Fighting malware on mobile devices has been the subject of numerous efforts. Most notable is the Google Bouncer system [Bou] for vetting apps before they get to users. This is in practice only a partial solution since there are many obfuscation techniques that can be employed to defeat both the static and dynamic analysis features of Bouncer. It has been demonstrated [Bou12] that it is quite trivial for an attacker to fingerprint the characteristics of the dynamic analysis of Bouncer. An adversary can write an app that triggers its malicious payload only when it detects that it is running on an actual device and not within the Bouncer testing and analysis platform.

Mobile antivirus

In addition to Google's efforts, several complementary static [EOMC11] and dynamic analysis [BSB10, Pro] techniques have been applied. Antivirus companies have also developed products for mobile systems that attempt to mimic the functionality of anti technology on the desktop. In February 2012, the German research institute AV-Test [AV-12] analyzed 41 anti-malware products for Android devices and found most failed to detect some malicious apps. The products only stati-

cally scan the application code and package information, without inspecting the runtime behavior of the apps. Indeed, antivirus software on a mobile device is only as privileged as any other app. The sandboxing mechanism that ensures the isolation of apps also prevents the antivirus software from actively monitoring an arbitrary app's behavior, such as its networking and file system access.

Application repackaging

Because runtime monitoring is necessary to detect when applications are acting maliciously, many solutions are implemented by repackaging application code with embedded custom monitoring software. This approach allows app behavior to be monitored without modifying the underlying operating system. To be practical, repackaging is best done without the need to decompile and reverse engineer the application code. However, only the Aurasium system [XSA12] implements this approach by "detouring" libc calls on Android and iOS. This is possible on all mobile platforms by detouring library calls during dynamic loading before an application starts running. Aurasium's design allows it to repackage arbitrary applications even when the code is obfuscated.

Alternative approaches on Android identify application calls to sensitive Java framework APIs and replace them with calls to their own detour functions. These approaches can therefore only handle apps written in Java and cannot monitor the parts of apps written in native C code. By repackaging target apps to introduce hooks that interpose on library calls and parsers to monitor inter-process communication, Aurasium is able to mediate virtually all API calls that apps make to the operating system. The interposition is implemented in native code and even wraps the Dalvik virtual machine itself, making it robust against arbitrary Java code even if it is loaded dynamically. The repackaging routine also ensures that Aurasium code is called to establish the sandbox before the target app gets control. Hence, any attempts by the app to load untrusted native code that could break out of the sandbox will always be detected.

Runtime monitoring

A number of academic efforts [EGC10, HHJ11, DSP11, NKZ10, BZNT11, BDD12] have focused on augmenting the Android platform with monitoring and policy enforcement mechanisms at different levels of the Android software stack. However, they have limited impact since they require significant changes in Android and have no plan to have these accepted by the operating system's maintainers. In contrast, SPADE [GT12] is able to track system-wide data provenance using the kernel auditing subsystem present in Android devices that use version 3.4 (or later) of the Linux kernel. Since it uses events recorded in the kernel, SPADE can monitor all activity including native code.

Virtualization

A far more ambitious effort uses virtualization to implement rigorous separation of trust domains, ranging from logical separation of private and public data on the device [SLA11] to running multiple instances of the operating system on the same device through the use of a hypervisor [GPHB11, LLL11, ADH11]. While this approach can effectively reduce the risk from mobile malware, it can only be adopted if the carriers and device manufacturers employ it. Further, the strong isolation may result in information sharing across partitions using mechanisms that are more vulnerable than local communication. For example, a calendar application may synchronize with a cloud service to provide a consistent view in each partition.

Any attempts by the app to load untrusted native code that could break out of the sandbox will always be detected.

Lessons learned

When assessing the state of the art of mobile security, there are many criteria that one might consider. Perhaps the most important criterion is the effectiveness of the solution. However, mobile platforms pose an additional challenge since they are not completely open; that is, the effectiveness of any solution can only be measured if it is widely adopted and deployed. Because of the various controls utilized by operating system providers, original equipment manufacturers (OEMs), and telecommunication carriers, the deployment of security solutions is all but certain.

Development

A significant challenge for creating security solutions is that they must rely on a root of trust. Even when the improvements are targeted at an open platform, such as the Linux kernel used by Android, completing the development is only the first step. The changes must then be accepted by the community. This requires engagement with a rapidly changing target. To understand the scale, consider that as of March 2013 there were close to 3,000 developers from over 400 companies contributing to the Linux kernel. About 20% of the code comes from individuals. Version 3.8.0 of the kernel received 7.38 changes every hour. The most prolific developer contributed close to 1,500 patches. The maintainer that signed off on the most patches accepted over 7,000 patches. Despite this, patches that result in a correct build, have been documented, and are written in the correct style will typically be accepted in two weeks [KH13].

When a developer sends a patch, it is reviewed by one of about 700 maintainers who are each responsible for specific drivers and files in the kernel. Either they push the patch back to the developer or they send it to one of nearly 100 subsystem maintainers. The kernel maintainer periodically opens a merge window during which the subsystem maintainers send all the changes that they have accepted. The code is merged by applying each patch and checking that the build still works. Companies like IBM and Intel submit patches for hardware when it is still in the design stage. By the time the hardware is released, the stable Linux kernel includes support for the

hardware and can be deployed immediately. By contributing the patches upstream, the companies also avoid bearing the cost of adapting the kernel to their hardware [KH13].

Deployment

While app vetting is deployed by official markets, none of the other security approaches are endorsed or deployed by mobile operating system providers. It is up to the carriers and eventually the users to decide which approach to adopt. Mobile antivirus applications that are available in the app markets simply check installed apps for known malware signatures. However, if deployed by carriers as a preinstalled system app, they can be run with sufficient privileges to automatically block and delete malicious apps without prompting the user for explicit approval. Application repackaging approaches can be deployed without modifications to the underlying operating system but are only available in enterprise stores. This is due to the fact that defining security policies for runtime monitoring of arbitrary apps is not practical. On the other hand, enterprise policies are easier to define. In addition, they can be applied across the set of all approved enterprise apps. All the other approaches require a modification to the underlying operating system. This incurs a significant cost for both the operating system providers and the carriers that must test devices prior to allowing them on their networks.

Cost

While the cost of app-level analysis and monitoring is low, the cost of modifying a mobile operating system to incorporate custom security solutions is prohibitive. It takes a device manufacturer about nine months to complete a quality assurance phase prior to a typical nine month sale period. Any modification in the operating system will increase the quality assurance period and reduces the shelf life of the device, reducing the manufacturer's and the carrier's profit margin. Virtualization-based approaches might seem less costly because the hypervisors are optimized for a specific processor. Additionally, the optimizations are done well before the devices are manufactured, typically as soon as the processor specification and simulator are available. However, paravirtualization is very costly if the goal is to isolate and sandbox all drivers. For every version of a mobile operating system, complete paravirtualization may take months. Previrtualization [MGS11] can be done efficiently and takes far less time to complete, but might impact the quality assurance phase, depending on the depth of previrtualization process.

Effectiveness

Many of the proposed mobile security approaches are limited in the scope of the threats that they can deal with. Solutions at lower levels in the software stack have greater coverage – virtualization-based approaches cover more threats than app-level approaches, for example. However, with breadth comes a lack of visibility into finer-grain application behavior. Consider the case of system call monitoring. When this is done for the entire device, many of the app-level semantics are lost. Common system calls such as read and write are too generic to be used to determine the correctness of the behavior of an application. Much additional processing is required to recover the semantics by analyzing the calling context. In general, the abstraction level of event monitoring on a device determines the effectiveness against specific threats.

Future directions

Given the tight control of current mobile platforms, the fragmentation of the market, and the challenge of deploying security solutions, it is necessary to think about novel approaches that provide innate security without drastic changes to the software stack and that avoid the typical adoption pitfalls. Paramount to the success of these new approaches is their support for legacy systems.

Trusted execution environments

Amongst the many proposed approaches to mobile security, perhaps hardware-based approaches have been the least explored. This is mainly due to the lack of a standardization of

trusted execution environments, such as ARM's Trust Zone. However, it is easy to imagine bootstrapping a chain of trust from hardware all the way up to an application. This opens up the opportunity to develop many security-critical applications, such as those used for interacting with banking systems or medical devices.

Sandboxing and virtualization

Because isolation is a fundamental security principle, it is necessary to adopt better sandboxing mechanisms. Paravirtualization is an approach that allows for the total isolation of drivers. This implies control of drivers, that is, all interactions with drivers occur through well-defined interfaces. This will allow the specification of security policies that can not only deny access to a driver, but can also state finer-grain changes, such as modifying its parameters, input, or output. Similarly, hardware manufacturers can advance the state of the art for virtualization by providing more controls. The A15 ARM core is a step in this direction.

Static previrtualization [MGS11] combines techniques from partial evaluation and link-time-optimization to specialize a software stack to a particular runtime context. By eliminating extraneous functionality, it removes potential vulnerabilities and can be used to reduce the attack surface of target applications.

Clean-slate approaches

Compartmentalization is not a sufficient requirement for securing mobile platforms. It is also necessary to control sensors so that access is granted only when necessary. Hardware-based controls will ensure that access only occurs through a non-bypassable mechanism. If software compartmentalization is used to implement such controls, they will not be as trustworthy.

Modern hardware architectures offer a number of covert channels such as direct memory access (DMA) to GPUs that has privileged visibility into buses and the sensitive data traversing them. A hardware sandboxing solution could be used to limit access to such resources. Depending on the architecture this may require hardware components to be updated to work with the sandboxing mechanisms.

CHERI [WNW12] is a research project that uses a revised hardware instruction set architecture (ISA) to better support software compartmentalization. It maps the Capsicum [WALK10] hybrid capability model into the CPU architecture, allowing fine-grained compartmentalization within process address spaces – while continuing to support current software designs. However, CHERI does not extend sandboxing to the driver level, though such an extension is planned.

Conclusion

Our goal has been to provide an overview of the current state of mobile security, describe the lessons learned during the course of projects that have focused on improving the area, and suggest some topics that future research, development, and deployment efforts may consider. In particular, we have highlighted the challenges encountered by the community, the range of attack surfaces that must be considered, and the variety of technical approaches that have been explored thus far. By identifying auspicious directions for further investigation, we hope to help prioritize the areas most in need of further analysis.

Acknowledgements

This work was funded under contract by the US Department of Homeland Security (DHS) Science and Technology (S&T) Directorate. The content is solely the product and responsibility of the authors and does not necessarily represent the official views of DHS.

References

[ADH11] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. Cells: a virtual mobile smartphone architecture. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, *SOSP'11*, pages 173–187, New York, NY, USA, 2011. ACM.

[AV-12] Test Report: Anti-malware solutions for android. Technical report, AV TEST: The Independent IT-Security Institute, Mar. 2012 – http://www.av-test.org/fileadmin/pdf/avtest_2012-02_android_anti-malware_report_english.pdf.

[BDD12] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. Towards taming privilege-escalation attacks on android. In NDSS'12, Feb 2012.

[Bou] Bouncer: Automated scanning of android market – http://googlemobile.blogspot.fr/2012/02/android-and-security.html.

[Bou12] Dissecting Android's Bouncer – http://blog.duosecurity.com/2012/06/dissecting-androids-bouncer/, June 2012.

[BSB10] Thomas Bläsing, Aubrey-Derrick Schmidt, Leonid Batyuk, Seyit A. Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In *5th International Conference on Malicious and Unwanted Software (Malware'2010)*, Nancy, France, France, 2010.

[BZNT11] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, *SPSM'11*, New York, NY, USA, 2011. ACM.

[DoD] http://www.bloomberg.com/news/2013-05-10/pentagon-plans-to-clear-apple-devices-for-network-use.html.

[DSP11] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S. Wallach. Quire: lightweight provenance for smart phone operating systems. In *Proceedings of the 20th USENIX conference on Security*, *SEC'11*, pages 23–23, Berkeley, CA, USA, 2011. USENIX Association.

[EGC10] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of OSDI'10*, Berkeley, CA, USA, 2010.

[EOMC11] 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.

[GPHB11] Kevin Gudeth, Matthew Pirretti, Katrin Hoeper, and Ron Buskey. Delivering secure applications on commercial mobile devices: the case for bare metal hypervisors. In *Proceedings*

of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM '11, pages 33–38, New York, NY, USA, 2011. ACM.

[GT12] Ashish Gehani and Dawood Tariq. SPADE: Support for provenance auditing in distributed environments. In *ACM/IFIP/USENIX 13th International Middleware Conference*, pages 101–120, 2012.

[HHJ11] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security, CCS '11*, pages 639–652, New York, NY, USA, 2011. ACM.

[Jun12] Juniper Mobile Security Report 2011. Unprecedented Mobile Threat Growth, Feb. 2012.

[KH13] Greg Kroah-Hartman. I don't want your code! Linux kernel maintainers, why are they so grumpy? In *Linaro Connect*, March 2013.

[LLL11] Matthias Lange, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. L4android: a generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM '11*, pages 39–50, New York, NY, USA, 2011. ACM.

[Loo11] Lookout Mobile Threat Report – https://www.mylookout.com/mobile-threat-report, Aug. 2011.

[MGS11] Gregory Malecha, Ashish Gehani, Natarajan Shankar, Bruno Dutertre, and Sam Owre. Previrtualization: Specializing software for security. Technical report, Computer Science Laboratory, SRI International, August 2011.

[NKZ10] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of ASIACCS '10*, New York, NY, USA, 2010. ACM.

[Pro] The Honeynet Project. Android reverse engineering virtual machine – http://www.honeynet.org/node/783.

[SLA11] Bugiel Sven, Davi Lucas, Dmitrienko Alexandra, Heuser Stephan, Sadeghi Ahmad-Reza, and Shastry Bhargava. Practical and lightweight domain isolation on android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM '11*. ACM, 2011.

[Tre13] http://www.neowin.net/news/android-malware-to-reach-1-million-cases-in-2013, March 2013.

[WALK10] Robert N. M. Watson, Jonathan Anderson, Ben Laurie, and Kris Kennaway. Capsicum: practical capabilities for unix. In *Proceedings of the 19th USENIX conference on Security, USENIX Security'10*, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[WNW12] Robert N.M. Watson, Peter G. Neumann, Jonathan Woodruff, Jonathan Anderson, Ross Anderson, Nirav Dave, Ben Laurie, Simon W. Moore, Steven J. Murdoch, Philip Paeps, Michael Roe, and Hassen Saidi. Cheri: a research platform deconflating hardware virtualization and protection. In *Runtime Environments, Systems, Layering and Virtualized Environments*, RESoLVE, 2012.

[XSA12] Rubin Xu, Hassen Saidi, and Ross Anderson. Aurasium: Practical policy enforcement for android applications. In *21st USENIX Security*, 2012.

About the Authors

Dr. Hassen Saidi, a senior computer scientist at SRI International, is an expert on malware analysis and reverse engineering. His PhD from the University of Joseph Fourier introduced predicate abstraction. He is the co-chair of the Mobile Platforms Fraud Working Group. He may be reached at hassen.saidi@sri.com.

Dr. Ashish Gehani of SRI International holds a BS (Honors) in Mathematics from the University of Chicago and PhD in Computer Science from Duke University. His research focuses on data provenance and security. He may be reached at ashish.gehani@sri.com.