Teaching Android Mobile Security

Jean-François Lalande
Valérie Viet Triem Tong
Pierre Graux
Guillaume Hiet
jean-francois.lalande@inria.fr
CentraleSupélec, Inria, Univ Rennes, CNRS, IRISA
Rennes, France

Wojciech Mazurczyk
Institute of Telecommunications
Warsaw University of Technology
Warsaw, Poland
wmazurczyk@tele.pw.edu.pl

Habiba Chaoui
National School of Applied Sciences, Ibn Tofail University
Kenitra, Morocco
habiba.chaoui@uit.ac.ma

Pascal Berthomé
INSA Centre Val de Loire, LIFO
Bourges, France
pascal.berthome@insa-cvl.fr

ABSTRACT

At present, computer science studies generally offer courses addressing mobile development and they use mobile technologies for illustrating theoretical concepts such as operating system, design patterns, and compilation because Android and iOS use a large variety of technologies for developing applications. Teaching courses on security is also becoming an important concern for academics, and the use of mobile platforms (such as Android) as supporting material is becoming a reasonable option. In this paper, we intend to bridge a gap in the literature by reversing this paradigm: Android is not only an opportunity to learn security concepts but requires strong pedagogical efforts for covering all the aspects of mobile security. Thus, we propose teaching Android mobile security through a two-dimensional approach. The first dimension addresses the cognitive process of the Bloom taxonomy, and the second dimension addresses the technical layers of the architecture of the Android operating system. We describe a set of comprehensive security laboratory courses covering various concepts, ranging from the application development perspective to a deep investigation of the Android Open Source Project and its interaction with the Linux kernel. We evaluated this approach, and our results verify that the designed security labs impart the required knowledge to the students.

CCS CONCEPTS

• **Applied computing** → **Education**; • **Security and privacy** → **Mobile platform security**; *Software security engineering*; **Software reverse engineering**.

This work has received a French government support granted to the COMIN Labs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287406

KEYWORDS

teaching, mobile, security

ACM Reference Format:
Jean-François Lalande, Valérie Viet Triem Tong, Pierre Graux, Guillaume Hiet, Wojciech Mazurczyk, Habiba Chaoui, and Pascal Berthomé. 2019. Teaching Android Mobile Security. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287406

1 INTRODUCTION

At present, Android programming is an integral part of most undergraduate studies in computer science. It was introduced as a complete knowledge area in 2013 (platform-based development) in the Curriculum Guidelines for Undergraduate Degree Programs in Computer Science [4]. The same year, security aspects were also introduced as a new knowledge area named "Information Assurance and Security." The Android platform can be used for a variety of learning purposes concerning operating system, wireless communications, advanced design pattern, etc. Skills in mobile programming are of significance as even web services generally develop a companion application that the user can install on a smartphone.

When addressing the security aspects of mobile computing, it is challenging to discuss concepts without referring to their implementation. Security concepts such as access control, authentication, or cryptography can still be presented as an independent course. However, the manner in which attackers attempt to bypass the implemented security countermeasures of modern smartphones should be presented with complete knowledge of the Android's internal aspects.

In this paper, we propose a set of labs intended to teach about threats and the security internals of Android applications and operating system. The approach is original in that it follows a progression based on the Bloom taxonomy [12] and works on the security aspects of the complete software stack of Android. We propose a variety of activities that enhance students' skills from those for developing simple applications to those for more complex activities such as analyzing vulnerable applications under attack or reversing packers. Additionally, the labs cover the different internal components of Android, which aids students to gain knowledge

of the internal classes of the Android runtime, the execution and compilation of Android applications, and the link with the operating system. These two dimensions—the student cognitive process and the coverage of the different levels of the Android software stack—aid students to better analyze mobile security problems.

The paper is organized as follows: Section 2 summarizes the state of the art in the learning of Android development and security. Section 3 presents our approach, and Section 4 provides the technical details of each lab. Section 5 summarizes the feedback we have collected. Section 6 concludes the paper.

2 RELATED WORK

Application Development and Operating Systems with Android. It is apparent that a pre-requisite for teaching mobile security is fluency in the underlying programming language and the main concepts of application development. For example, Tigrek et al. [21] propose a minimal course for first year engineering students to discover Android programming. Among the presented principles, "How Without Why" (program based on tutorials) combined with "Just Enough Java" (learn the essential Java patterns) are effective for learning to develop an application, although is not sufficient for understanding the security impact of a selected implementation [1]. In Bloom's revised taxonomy of educational objectives [12], these two pieces of procedural knowledge are used to distinguish between "knowledge of subject-specific skills" and "knowledge of criteria for determining when to use appropriate procedures". Additionally, we consider that in the structure of the cognitive process (remember, understand, apply, analyze, evaluate, and create) the three last cognitive processes require a deep understanding of the internal aspects of mobile application and the underlying operating system.

The Android framework can also be used as supporting material to teach more general software engineering [5, 20] or operating system [2] concepts. Notwithstanding whether these works are effective for a better understanding of the concepts, most of the presented material does not describe the Android system with adequate details to aid the investigation of the security aspects of mobile devices.

Mobile Security. A few studies use Android as a platform for experimenting with general security notions, e.g., filtering traffic in firewalls [23] or countering Denial of Service attacks [22]. However, they do not address the specificities of mobile systems and applications.

A few previous works [10, 15, 25] have described the particularities of teaching security aspects of mobile applications and platforms. Nevertheless, in most cases, only foundational aspects that are relevant for Android developers are covered.

The approach of Guo *et al.* [10] is one of the works closest to our proposal. They have designed a labware of seven modules, which illustrates the concepts of mobile security and privacy. The learning approach is based on two main concepts. First, the course uses an attack/defense lab, where the students are required to implement attacks prior to implementing the corresponding protection mechanisms. Such an approach aids the students to better comprehend the security issue and to implement more effective protections. Secondly, the provided material (slide, tutorials) and the content of the labs can be used directly on real smartphones, enhancing the confidence of the students in their skills. Each lab intends to develop skills of protection against one type of threat. The fundamentals of Android security are introduced in different labs, such as those on the use of permissions or cryptographic API. Advanced threats and prevention mechanisms, such as buffer overflows or obfuscation, are discussed in a lab entitled "Secure Mobile App Development," which was under development at the time of the article writing.

In our opinion, these works lack progression in the Bloom taxonomy and do not cover the different levels of the Android software stack. The next section develops this concept before the description of the labs in Section 4.

Table 1: Attacks targeting different Android components

Components	Examples of attack	CVE
Applications	Android ransomware, spyware, adware	
AOSP classes	OpenSSLX509Certificate vulnerability [17]	2015-3825
DVM & ART	DoS via an unspecified Dalvik function	2009-3698
AOSP services	Janus: Hiding Payload & Bypass signatures	2017-13156
Kernel	Towelroot: Futex Requeue Kernel Exploit	2014-3153

3 LEARNING MOBILE SECURITY

A deeper understanding of all the security components of Android requires comprehension of the whole stack of technologies that is used by the Android Open Source Project (AOSP) [6, 9]. Describing security APIs from the developer's perspective can be a starting point for discovering features such as account management, permissions, and cryptographic primitives. Nevertheless, most of the security features are provided by the operating system, and explaining them requires sufficient knowledge of operating systems, runtime environments, and networks. Examples of such features are the installation process of applications, storage of credentials, role of SELinux, and update process of the operating system.

Additionally, students should comprehend the heterogeneous nature of the attacks likely to target Android devices. We have presented a few examples of vulnerabilities in Table 1 that illustrate the diversity of the likely threats. The kernel is the most challenging component to attack, and a representative vulnerability is the Futex vulnerability[1], which enables, under certain conditions, the execution of arbitrary code in the kernel mode and to root the phone. On top of the kernel, the Android services can be exploited, *e.g.*, the process that installs or updates applications. An example of such a threat is the Janus exploit that abuses a vulnerable signature verification implementation. Similar attacks target the direct environment of the malicious application, *i.e.*, the Dalvik virtual machine or the ART library that implements security verifications for the running application. Moreover, the attacker can attempt to abuse the SDK itself by identifying vulnerable classes [17] with controllable side-effects. Finally, and this constitutes a significant part of the attacks, the attacker does not exploit any vulnerability but performs malicious activities using the requested permissions. Examples of such applications are ransomware, cryptominers, and remote administration tools.

Taking these remarks into consideration, we describe the structure of the proposed labs and detail their content, in the next section.

[1] https://tinyhack.com/2014/07/07/exploiting-the-futex-bug-and-uncovering-towelroot

Table 2: Proposed security activities in Bloom taxonomy of cognitive process structure [12]

Soft. components \ Cognitive process	Remember	Understand	Apply	Analyze	Eval.	Create
Applications	DEV app development	MAL malware reverse				PROJ
AOSP classes		BANK banking app reverse	COV covert channels	CLASS vulnerable class loader		
DVM & ART				PACK reverse packers		
AOSP internals	INST compile, flash		MEM memory forensic	KERN ROP programming		
Kernel						

Table 3: Knowledge chapters for the DEV lab

Chapter 1
Application's architecture [13, 26]
Designing graphical interfaces
Messaging components
Chapter 2
Concurrency and synchronization
Connectivity and sensors
Chapter 3
Security [6, 9]
Wear OS [8]
Firebase Cloud Messaging

4 ANDROID SECURITY LABS

4.1 Security Labs Design

We have developed several labs[2] which cover different Android components involved in the understanding of mobile security. We present these labs in Table 2, following Bloom's revised taxonomy of educational objectives [12]. Students should follow these labs from left to right in the table, eventually completing the sequence with a security project (PROJ) that corresponds to the deepest cognitive process of the taxonomy (Create).

The number of hours depends of the level of knowledge teachers intend to obtain. We distinguish three levels: level 1 (~2h) provides fundamental knowledge or focuses on a particular aspect; level 2 (~5h-8h) enables the student to cover more notions and practice with real devices; level 3 (~20h) aids the student to cover a large variety of notions or to work on a particular one with important software development. Most of the labs use level 1 or 2 (e.g., INST and MAL); however, a few of them (e.g., DEV and COV) can use level 3 or can be adapted as a security project (PROJ).

The first two labs (Remember process) train the students to develop Android applications (DEV) and to flash a custom Android image on a smartphone (INST). Such labs are typically observed in other available approaches [2, 20, 21]. Then, two labs enhance the students in the understanding of benign (BANK) and malicious applications (MAL) by reversing their bytecode. This aids the studying of attacks at the application level and their associated countermeasures (debugger detection, obfuscation, etc.) that can be used both by malicious and legitimate applications.

The subsequent labs correspond to the "Apply" step in the taxonomy. Students are encouraged to implement attacks and countermeasures with their previously acquired knowledge. For this purpose, we propose two use cases. The first one (COV) is proposed to defeat the Android access control system by implementing a covert channel between two applications that do not have permission to interact with each other. The second one (MEM) is proposed to exploit a dump of the volatile memory to recover certain confidential information. The likely benefit of these labs is to compel the students to obtain deeper knowledge of the low-level components of Android, *i.e.*, the leakage induced by the operating system and its primitives (COV) and the Dalvik (or Art) data structures used to represent the Java classes of the applications in volatile memory (MEM).

[2] https://gitlab.inria.fr/jlalande/teaching-android-mobile-security

Contrary to the previous labs, wherein students are completely guided while they practice, the three subsequent labs require the performance of security analysis (Analyze process). We provide the symptoms of certain issues to the students, and we expect an investigation and eventually certain remediation. In the first lab (CLASS), students are required to analyze an application that uses a vulnerable class loader. The second lab (PACK) proposes to analyze a malicious application obfuscated with packing techniques. The third lab (KERN) focuses on low level vulnerabilities by performing ROP attacks using gadgets of the kernel.

Note that no lab is classified in the "Evaluate" cognitive process as the distinction between "Analyze" and "Evaluate" is marginal. In particular, in [4], only three levels of mastery is proposed (familiarity, usage, assessment). Thus, we use only four levels of the Bloom taxonomy for classifying our labs; additionally, we propose various security projects (PROJ). They are designed to explore a specific subject with a large number of hours (level 3 or higher) for a group of students and may be linked to research activities.

4.2 DEV Lab: Android Development

The DEV lab provides the pre-requisite for understanding the fundamentals of Android development. We classify this lab in Bloom's taxonomy in the "Remember" process as we do not intend to train software developers and rather intend to teach the fundamentals for discussing security aspects. The topics covered depend on the amount of time allocated, and we distinguish three chapters of knowledge, as presented in Table 3.

Learning outcomes. At the end of the lab, we expect the students to comprehend the architecture and deployment of an Android application. The students should be capable of developing an application that communicates with a remote server, using REST API.

4.3 INST Lab - Compiling, Modifying, Flashing

Numerous other laboratories presented in this paper are required to be able to setup a clean image of the smartphone. As indicated by Guo et al. [10], students appreciate having a real device to experiment with while performing development or hacking activities. Moreover, it is occasionally more difficult to work with an emulator, for example to play with the different sensors, or to pair an emulated smartphone with an emulated Wear watch, which may require the installation of the Google APIs with an active Google account.

The INST lab (level 1) includes the compilation of a complete AOSP distribution for a Google Nexus 5 or 5X or Sony Xperia X smartphone and then the flashing of the compiled images. Additionally, the Nexus models offer the capability to debug the kernel output using serial headphone jack. For that purpose, we constructed a small serial debugger that aids students involved in advanced labs, where the kernel is modified and is likely to crash during runtime.

Learning outcomes. Students are able to customize the source of an operating system and install it on a device.

4.4 MAL Lab - Malware Reverse Engineering

The MAL lab consists in reversing several types of malware using different open source tools. The pedagogical goal of the lab is to demonstrate that each type of malware requires a custom reverse process. The lab consists of multiple exercises. Thus, the required number of hours can be adapted from level 1 to 3. In Bloom's taxonomy, we place this lab in the "Understand" process because we provide the students with the entire set of instructions for reversing the malware. We present the two most representative ones.

Programming an antidote for a ransomware. The students install a ransomware on a provided smartphone. They observe that the image files are encrypted by the malware. Then, the students explore the code using BytecodeViewer[3] or Jadx[4] in order to identify the service that encrypts the images. The code analysis reveals that the AES ciphering key used by the ransomware is a constant. Thus, after locating the deciphering code, the students modify the malicious application using to a bytecode editor so that it calls the deciphering code. Finally, they use this modified application to recover the images on the smartphone.

Observing spyware. The students are first required to reverse the spyware and locate the code that extracts personal data (accounts, phone number, IMEI) and the code that transmits the data to a remote server, whose domain name is identified. for the sake of simplicity. We also propose the observation of the leaked data in network traces as part of the lab in order to confirm the static analysis. To achieve this goal, the students setup a local web server, use the ngrok website[5] to create a public URL of the form x.ngrok.io, and link this URL to the local web server. Then, using a provided tool based on Soot[6], the students substitute the domain name used by the malware with this URL. Executing this new version of the malware enables one to capture the first requests that are responsible for leaking user's data.

Learning outcomes. Students comprehend the different types of threat that security analysts confront. They discover the process of reverse engineering with three approaches. This emphasizes the fact that security analysts have to adapt their methodology to the nature of the threat. At this stage, the investigation process is specified; however, students understand that a custom process should be designed for each malware.

[3]https://bytecodeviewer.com
[4]https://github.com/skylot/jadx
[5]Ngrog is a tunneling service from a public URL to a local server: https://ngrok.com
[6]http://sable.github.io/soot/

4.5 BANK Lab - Banking Application Reverse

This lab consists in reversing the authentication activity of a real banking application. This lab is completely guided, which places it in the "Understand" process of the taxonomy and aids to fit its duration to level 2.

The students are required to comprehend the protocol that is used by the application to authenticate the user to a remote server. The banking application receives a challenge from the server, which is used to generate a virtual keyboard. This permits the application to obfuscate the entered password.

First, students unpack the application and uncompile the bytecode into Java source code using Jadx. This phase aids them to identify the activities and classes involved in password encoding. Second, students observe the evolution of the application state. Using AndBug[7], students can put breakpoints on specific bytecode instructions. In particular, it aids them to identify the code that transmits the encoded password to the server. They also install a counterfeit certificate in the smartphone to bypass the verification of the remote server's authenticity. This permits them to monitor the network traffic using the Burp Suite proxy[8]. Finally, the students precisely identify the Java objects that encode the password and that are transmitted to the server when the user attempts to authenticate. By capturing both the challenge received from the server and the encoded password transmitted by the application, the students can recover the password.

Learning outcomes. Students comprehend the countermeasures used by applications for critical parts such as the authentication phase. They train themselves to bypass these countermeasures as a real attacker would. Working on attack design aids the understanding of the limits of current security implementations.

4.6 COV Lab - Developing Covert Channels

The COV lab (level 2 or 3) consists in developing a covert channel between two Android applications [14] in order to bypass the enforced security policy.

We provide two applications App1 and App2 to the students. App1 collects certain sensitive data of a user but does not have the permissions to use the network. App2 cannot access confidential data but has the necessary permissions to use the network. Those applications cooperate to leak the data. To achieve this, App1 establishes a local covert channel with App2 which is based on the remaining free space in the flash memory. This can encode the secret data bits if the sender (App1) creates and deletes large files. Snippets of code are provided to the students to aid them to develop the covert channel inside an asynchronous task. Moreover, countermeasures against such threats are discussed and analyzed. If time permits, certain countermeasures can be implemented using papers of the literature [14, 24].

Learning outcomes. By completing this lab, the students become aware of the threat posed by covert channels and the means by which an enforced security policy can be bypassed. Students can also discover advanced detection techniques.

[7]https://github.com/swdunlop/AndBug
[8]https://portswigger.net/burp

4.7 MEM Lab - Memory Dump Forensic

This lab (level 2) has been inspired by studies on recovering credentials from volatile memory [3, 19]. It is highly challenging for Java applications to clean their memory after allocating certain objects even after rebooting the device [16]. Thus, we have designed a lab wherein students inspect a dump of the memory obtained with Lime[9]. This dump is achieved after displaying a simple application with two `EditText` objects, which simulates an authentication activity. The goal of the lab is to capture the content of the two `EditText` objects from the dump.

Then, students should inspect the memory dump using the Volatility framework[10]. After identifying the Linux process corresponding to the Dalvik VM, we provide extensions for inspecting the heap of the virtual machine based on the work of Hilgers et al. [11]. Using Python scripts, students can manipulate classes and objects and their implementation at the C programming language level. Students are invited to write the script that enumerates all the classes and to identify the ones that extends `EditText`. This aids them to retrieve the credential from the memory dump.

Learning outcomes. Students comprehend the likely leaks induced by the memory management of Java-like virtual machines, Dalvik and Art. They are capable of conducting a simple forensic of a memory dump, which is of interest to digital investigators.

4.8 CLASS Lab - Vulnerable Class Loader

The CLASS lab (level 2) places the students in a situation wherein an attacker has compromised the remote server used by the application. We ask students to analyze and solve the situation following Bloom's taxonomy. This learning approach has already been investigated for learning web development in a secure manner [18]. In our lab, the vulnerable application loads a few classes from the remote server that is supposed to have been compromised by the attacker. The students have access to the source code of the first activity although not to the sources of the remote classes. They are required to investigate different attacks and if feasible, implement countermeasures.

The vulnerable application is composed of a regular activity that dynamically loads other activities from the remote server. The vulnerabilities are twofold. First, the application uses a custom class loader that is ineffectively implemented because it systematically loads classes from the remote resources rather than first asking the hierarchy of class loaders. Thus, the classes from the Android runtime such as `Checkbox` or `LinearLayout` can be overwritten by the downloaded activities. This can be used by the attacker to take control of the main activity. Secondly, the first activity stores certain confidential data in a static field. The attacker can access it later from the loaded activity, which is an unintentional leak of information. To disable those attacks, the students are required to patch the class loader and use an instance field rather than the static field.

Learning outcomes. Students should be capable of conducting a complete investigation of an application connected to a compromised server. Tthey should understand that an implementation can contain vulnerabilities if the developer does not comprehend the mechanisms induced by the used language or the operating system.

4.9 PACK Lab - Packers

The PACK lab (level 2) consists in reversing the MAL lab ransomware that are now obfuscated with different packing techniques. The packer uses native code to unpack the payload at runtime. The obfuscated methods are decoded immediately before being called and are re-encoded when they return. This lab requires a deep analysis to determine how the obfuscated code can be retrieved.

First, the students execute the applications in order to observe that they behave as those of the MAL lab. Then, they use BytecodeViewer or Jadx to inspect a specified malware sample and determine that the code of certain methods is empty. Using IDA Pro[11], the students are required to discover how the native code obfuscates the Java methods. At this stage of the lab, it is feasible to retrieve the original code of the malware sample.

However, additional ransomware samples using different packing algorithms are provided to the students. Consequently, new static analysis of each packing algorithm would be required, and therefore, we propose to switch to a dynamic approach. The students are guided to modify the code of AOSP: The Android VM is modified to dump the code of each method before executing them. We provide the most challenging parts of the modifications to the students to aid them to complete the lab. Based on the INST lab, the students flash the Sony Xperia X smartphone with the modified Android system. This permits them to automatically reverse all the samples.

Learning outcomes. Students comprehend and analyze the work achieved by a packer using ciphering algorithms. They also comprehend the benefit of a solution based on a combination of a static analysis and a dump that is performed during execution.

4.10 KERN Lab - Kernel ROP Attacks

The KERN lab (level 2 or 3) goes deeper into the security aspects related to Android's kernel. In this lab, we provide an emulator with a kernel containing a vulnerable driver. This driver can be exploited with a buffer overflow vulnerability that enables the execution of an arbitrary short payload. Because the kernel implements a `W xor X` policy, students are required to perform a ROP attack with gadgets available in the kernel. For achieving this step, students use ROPgadget[12] to search for gadgets in the `vmlinuz` file. The "Hello World" example consists in printing a message in the kernel logs. Students should learn to call the `printk` kernel function using several gadgets. If time permits, more complex attacks can be achieved to spy the memory of a targeted process.

Learning outcomes. This lab illustrates the importance of the security of all Android components. It trains students to perform attacks using multiple vectors including attacking the system itself.

The material associated to these labs can be found online at https://gitlab.inria.fr/jlalande/teaching-android-mobile-security

[9] https://github.com/504ensicsLabs/LiME
[10] https://www.volatilityfoundation.org/
[11] https://www.hex-rays.com/products/ida/
[12] https://github.com/JonathanSalwan/ROPgadget

Table 4: Evaluation of progress (δ) and average mark after labs (m) for each type of lab

	Attended lab:	**DEV**		**INST**		**MAL**		**MEM**		**COV**		**CLASS**	
	Nb of students:	$n = 27$		$n = 7$		$n = 28$		$n = 3$		$n = 11$		$n = 11$	
Questions	Scoring	δ	m	δ	m	δ	m	δ	mark	δ	m	δ	m
Are you able to develop an Android application that interacts with an HTTP server?	**DEV**	**+1.74**	**3.67**	+0.14	2.14	+0.75	2.64	+0.00	1.67	+0.18	2.36	+1.36	2.73
Are you able to compile and/or flash an Android distribution (e.g. AOSP)?	**INST**	+0.93	2.59	**+2.57**	**4.14**	+0.82	2.71	+0.00	1.00	+0.18	2.09	+0.55	1.91
Are you able to reverse Android malware?	**MAL**	+0.56	1.74	+1.29	2.29	**+1.46**	**2.96**	+0.00	1.00	+0.18	1.82	+0.55	1.73
Are you able to perform a forensic analysis of a dump of volatile Android memory?	**MEM**	+0.56	1.78	+0.86	1.86	+0.86	2.11	**+2.00**	**3.00**	+0.00	1.55	+0.18	1.27
Are you able to develop a covert channel to hide communication between two Android applications?	**COV**	+0.52	1.74	+0.00	1.00	+1.00	2.14	+0.00	1.00	**+2.45**	**3.82**	+0.36	1.55
Are you able to comprehend how a vulnerable class loader can be exploited and propose some patch?	**CLASS**	+0.44	1.59	+0.29	1.43	+0.96	2.21	+0.33	1.33	+0.09	1.64	**+0.91**	**2.27**

1 = Unknown – 2 = Discovering – 3 = Intermediate – 4 = Good knowledge – 5 = Advanced

5 EVALUATION

5.1 Audience

The presented labs have been used in several undergraduate programs in eight universities and engineering schools of three countries. We did not have the opportunity to play all the labs for the same pool of students. Nevertheless, multiple combinations of two or three labs have been tested. We have also used these labs for research summer schools and tutorials at international conferences for illustrating technical aspects of mobile security. Nevertheless, at the time of writing, the goal of this work is also to introduce the whole sequence of labs in the master degree curriculum, where security aspects have an important position.

5.2 Evaluation Design

We designed a survey for asking students to evaluate their capabilities with regard to the labs they attended. The concept was to ask the students to evaluate themselves regarding only one lab in order to measure the correlation between the lab they followed and the skill evaluated. If they followed two labs, we asked them to evaluate themselves with respect to the lab with the highest number of hours. The survey was sent in spring 2018 to the students we had taught between 2014 and 2018. From among the approximately 200 students, we obtained responses from 87.61% followed the lab a few months ago, 19% a year ago, 14% two years ago, and the remaining 6% over two years ago.

Table 4 presents the obtained results. The left column recalls the asked questions which correspond to the pedagogical goal of a lab. On the right, student evaluations have been reported in the column corresponding to the name of the lab they followed. With such a presentation of the results, the correlation should occur on the diagonal of the table. For evaluating each question, we used the following marking system inspired by the work of Campbell et al. [5]: 1) Unknown (No trace in my memory); 2) Discovering (I recall some of the content); 3) Intermediate (I understood most of the content); 4) Good knowledge (I am able to do the lab again, without a supervisor and with the help of documents); 5) Advanced (I can reuse my knowledge in another use case).

For each question, a student is asked to rate himself/herself "before" and "after" the lab, which enables the computation of a differential score corresponding to the difference in the mark before and after the lab. This provides more information than the raw mark does with regard to whether the students have previously followed courses on related topics. It enables the evaluation of the progress of the students irrespective of their starting point. In Table 4, m corresponds to the average of the marks "after the lab" (from one to five) and δ corresponds to the average of the differential marks.

5.3 Evaluation Results

The correlation between the labs and the evaluated skills is evident: for each lab, the highest scores correspond to the pedagogical goal of the lab. The improvement in student's skills on the diagonal is an average of +1.85, which is encouraging as it is approximately equivalent to shifting from "Discovering" to "Good knowledge." Only the CLASS lab achieves an improvement of +0.91. This can be explained by the level of difficulty of the lab, which corresponds to the Analyze phase of the taxonomy and requires substantial effort of investigation by the students. The CLASS lab also achieves a progression of +1.36 for the DEV lab. This can be explained by the fact that the CLASS lab involves HTTP requests.

The self-evaluation of skills by the students yielded an average mark of 3.31 on the diagonal. The marks of the MAL and MEM labs are approximately equal to 3, which is a reasonable result considering the technical challenge associated with them. The COV labs achieved the highest mark (3.82). We conjecture this to be a result of the high number of allocated hours for this lab. More simple activities achieve higher results (DEV, INST) as students feel comfortable when performing the labs.

The results confirm that the labs are aligned with their pedagogical goal. The marks reveal that students tend to evaluate their skills between intermediate and reasonable knowledge levels. Indeed, only years of practice can provide further knowledge.

6 CONCLUSION

This paper explores several technical aspects with regard to the learning of Android security. Based on Bloom's taxonomy, we designed various labs that aid students to learn more deeply about the internal aspects of Android. By programming attacks and studying malware or vulnerable applications, students are trained to analyze complex security problems. This pedagogical design aids them to improve their skills in mobile security by providing a deeper understanding of the complexity of the software components used by Google to design its operating system.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 289–305. https://doi.org/10.1109/SP.2016.25

[2] Jeremy Andrus and Jason Nieh. 2012. Teaching operating systems using android. In *43rd ACM technical symposium on Computer Science Education*. ACM Press, Raleigh, North Carolina, USA, 613–618. https://doi.org/10.1145/2157136.2157312

[3] Dimitris Apostolopoulos and Giannis Marinakis. 2013. Discovering authentication credentials in volatile memory of Android mobile devices. In *12th IFIP Conference on e-Business, e-Services, e-Society*. Athens, Greece, 178–185. https://doi.org/10.1007/978-3-642-37437-1_15

[4] Muhammad Rizwan Asghar and Andrew Luxton-Reilly. 2018. Teaching Cyber Security Using Competitive Software Obfuscation and Reverse Engineering Activities. In *49th ACM Technical Symposium on Computer Science Education - SIGCSE '18*. ACM Press, Baltimore, MD, USA, 179–184. https://doi.org/10.1145/3159450.3159489

[5] Jennifer Campbell and Anya Tafliovich. 2015. An Experience Report: Using Mobile Development To Teach Software Design. In *46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*. ACM Press, Kansas City, MO, USA, 506–511. https://doi.org/10.1145/2676723.2677307

[6] Nikolay Elenkov. 2014. *Android Security Internals: An In-Depth Guide to Android's Security Architecture*. No Starch Press.

[7] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *9th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Vancouver, BC, Canada, 393–407.

[8] Andrey Esakia, Shuo Niu, and D. Scott McCrickard. 2015. Augmenting Undergraduate Computer Science Education With Programmable Smartwatches. In *46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*. ACM Press, Kansas City, MO, USA, 66–71. https://doi.org/10.1145/2676723.2677285

[9] Sheran Gunasekera. 2012. *Android Apps Security*. Apress.

[10] Minzhe Guo, Prabir Bhattacharya, Ming Yang, Kai Qian, and Li Yang. 2013. Learning mobile security with android security labware. In *44th ACM technical symposium on Computer science education - SIGCSE '15*. ACM Press, Kansas City, MO, USA, 675–680. https://doi.org/10.1145/2445196.2445394

[11] Christian Hilgers, Holger Macht, Tilo Muller, and Michael Spreitzenbarth. 2014. Post-Mortem Memory Analysis of Cold-Booted Android Devices. In *2014 Eighth International Conference on IT Security Incident Management & IT Forensics*. IEEE Computer Society, Munster, Germany, 62–75. https://doi.org/10.1109/IMF.2014.8

[12] David R Krathwohl. 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice* 41, 4 (2002), 212–218. https://doi.org/10.1207/s15430421tip4104_2

[13] Jonathan Levin. 2015. *Android Internals: A Confectioner's Cookbook*.

[14] Claudio Marforio, Hubert Ritzdorf, Aurélien Francillon, and Srdjan Capkun. 2012. Analysis of the communication between colluding applications on modern smartphones. In *28th Annual Computer Security Applications Conference*. ACM Press, Orlando, Florida, USA, 51–60. https://doi.org/10.1145/2420950.2420958

[15] Matthew Neis, Vincent Cefalu, and Ankur Chattopadhyay. 2018. Developing a Unique Android App-driven Nifty Middle-School Educational Module on Mobile Security for Driving Basic Information Security Awareness and Generating Interests in Cybersecurity. In *49th ACM Technical Symposium on Computer Science Education - SIGCSE '18*. ACM Press, Baltimore, MD, USA, 1081–1081. https://doi.org/10.1145/3159450.3162243

[16] Christoforos Ntantogian, Dimitris Apostolopoulos, Giannis Marinakis, and Christos Xenakis. 2014. Evaluating the privacy of Android mobile applications under forensic analysis. *Computers & Security* 42 (may 2014), 66–76. https://doi.org/10.1016/j.cose.2014.01.004

[17] Or Peles and Roee Hay. 2015. One Class to Rule Them All: 0-Day Deserialization Vulnerabilities in Android. In *9th USENIX Workshop on Offensive Technologies*. Washington, WA, USA, 1–12.

[18] Michael Sonntag. 2013. Learning security through insecurity. In *2nd International Conference on E-Learning and E-Technologies in Education*. Lodz, Poland, 143–148. https://doi.org/10.1109/ICeLeTE.2013.6644363

[19] Pasquale Stirparo, Igor Nai Fovino, and Ioannis Kounelis. 2013. Data-in-use leakages from Android memory - Test and analysis. In *9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 718–725. https://doi.org/10.1109/WiMOB.2013.6673433

[20] Kelvin Sung and Arjmand Samuel. 2014. Mobile application development classes for the mobile era. In *2014 conference on Innovation & technology in computer science education*. ACM Press, Uppsala, Sweden, 141–146. https://doi.org/10.1145/2591708.2591710

[21] Seyitriza Tigrek and Mohammad Obadat. 2012. Teaching smartphones programming using (Android Java): Pedagogy and innovation. In *2012 International Conference on Information Technology Based Higher Education and Training*. IEEE Computer Society, 1–7. https://doi.org/10.1109/ITHET.2012.6246039

[22] Zouheir Trabelsi, Mohammed Al Matrooshi, and Saeed Al Bairaq. 2016. A Smartphone App for Enhancing Students' Hands-on Learning on Network and DoS Attacks Traffic Generation. In *17th Annual Conference on Information Technology Education*. ACM Press, Boston, MS, USA, 48–53. https://doi.org/10.1145/2978192.2978229

[23] Zouheir Trabelsi, Mohammed Al Matrooshi, Saeed Al Bairaq, Walid Ibrahim, and Mohammad M. Masud. 2017. Android based mobile apps for information security hands-on education. *Education and Information Technologies* 22, 1 (jan 2017), 125–144. https://doi.org/10.1007/s10639-015-9439-8

[24] Marcin Urbanski, Wojciech Mazurczyk, Jean-Francois Lalande, and Luca Caviglione. 2017. Detecting Local Covert Channels Using Process Activity Correlation on Android Smartphones. *International Journal of Computer Systems Science and Engineering* 32, 2 (March 2017).

[25] Xiaohong Yuan, Kenneth Williams, Scott McCrickard, Charles Hardnett, Litany H. Lineberry, Kelvin Bryant, Jinsheng Xu, Albert Esterline, Anyi Liu, Selvarajah Mohanarajah, and Rachel Rutledge. 2016. Teaching mobile computing and mobile security. In *IEEE Frontiers in Education Conference*. IEEE Computer Society, Erie, PA, USA, 1–6. https://doi.org/10.1109/FIE.2016.7757365

[26] Karim Yaghmour. [n. d.]. *Embedded Android: Porting, Extending, and Customizing*. O'Reilly Media, Inc.