Data Protection Labware for Mobile Security

Hossain Shahriar[1(✉)], Md Arabin Talukder[1], Hongmei Chi[2], Mohammad Rahman[3], Sheikh Ahamed[4], Atef Shalan[5], and Khaled Tarmissi[6]

[1] Kennesaw State University, Kennesaw, GA 30144, USA
{hshahria, mtalukdl}@kennesaw.edu
[2] Florida A&M University, Tallahasse, FL 32307, USA
hongmei.chi@famu.edu
[3] Florida International University, Miami, FL 33174, USA
marahman@fiu.edu
[4] Marquette University, Milwaukee, WI 53233, USA
shiekh.ahamed@marquette.edu
[5] Alderson Broaddus University, Philippe, WV 26416, USA
shalanm@ab.edu
[6] Umm Al Qura University, Mecca, Kingdom of Saudi Arabia
kstarmissi@uqu.edu.sa

Abstract. The majority of malicious mobile attacks take advantage of vulnerabilities in mobile applications, such as sensitive data leakage via inadvertent or side channel, unsecured sensitive data storage, data transmission, and many others. Most of these mobile vulnerabilities can be detected in the mobile software testing phase. However, most development teams often have virtually no time to address them due to critical project deadlines. To combat this, the more defect removal filters there are in the software development life cycle, the fewer defects that can lead to vulnerabilities will remain in the software product when it is released. In this paper, we provide details of a *data protection* module and how it can be enforced in mobile applications. We also share our initial experience and feedback on the module.

Keywords: Mobile software security · Android · Data protection · Labware · SSL

1 Introduction

Despite the great need for mobile professionals and existing efforts in mobile security is a relatively weak and is not well represented in the computing curriculum. The challenges include scarce dedicated staff and faculty in this field and the excessive time needed for developing course materials and hands-on projects.

The majority of malicious mobile attacks take advantage of vulnerabilities in mobile applications, such as sensitive data leakage via inadvertent or side channel, unsecured sensitive data storage, data transmission, and many others. Most of these mobile vulnerabilities can be detected in the mobile software testing phase. However, most development teams often have virtually no time to address them due to critical

G. Wang et al. (Eds.): SpaCCS 2019, LNCS 11611, pp. 183–195, 2019.
https://doi.org/10.1007/978-3-030-24907-6_15

project deadlines [3]. To combat this, the more defect removal filters there are in the software development life cycle, the fewer defects that can lead to vulnerabilities will remain in the software product when it is released. More importantly, early identification of defects during implementation is better than taking corrective action after software release [4]. Many development professionals lack the necessary secure knowledge and skills during development stage [2].

As more schools develop teaching materials for mobile application development, there is a proportional growth in the need for educational activities promoting mobile security education in the development and security quality assurance phase [4]. However, mobile security is a relatively weak field and is poorly represented in most schools' computing curriculum.

As part of Secure Mobile Software Development (SMSD) [1] project, we are currently developing capacity to address the lack of pedagogical materials and real-world learning environment in secure mobile software development through effective, engaging, and investigative approaches. We developed a collection of eight transferrable learning modules with companions hands-on labs on mobile coding (e.g., data sanitization for input validation, secure sensitive data storages, secure inter-activity communication), which can be integrated into existing undergraduate and graduate computing classes that will be mapped to ISA KAs proposed in CS curricula 2013 to enhance the student's secure mobile software development ability.

In this paper, we provide details of one of the developed modules: data protection. We share our experience of student feedback on the learning module and reflection.

This paper is organized as follows. Sections 2 discusses the developed module in details. Section 3 provides the survey outcome of the learning module and initial feedback from classroom. Section 4 discusses related work. Finally, Sect. 5 concludes the paper.

2 Data Protection

2.1 Data Stability During Client-Server Communication

Socket programming SSL (Secure Socket Layer) in android can be used to preserve data for being leaked to the intruders by using tools like tcpdump and Wireshark. SSL provides encryption and decryption mechanism for the insurance of data. A server runs continuously to listen to the connection among clients and a client will initialize a connection with the server. Key and certificate play a valuable role for the shelter of data from malicious attempts. The procedure can be followed as depicted in Figs. 1 and 2.

The keytool command can be used to generate and self-assign a private key into a certificate. Preserve the certificate in a "keystore" file and export it to the client device for authentication. SSL server socket and client socket algorithm with the inclusion of keystore file is required to communicate between the server and the client, and vice-versa. Android requires Bouncy Castle certificate that could be added by importing JAR (bcprov-jdk16-145.jar). Two-way authentication requires a key generation for the client as well. An example of certificate creation is given in Figs. 3 and 4.

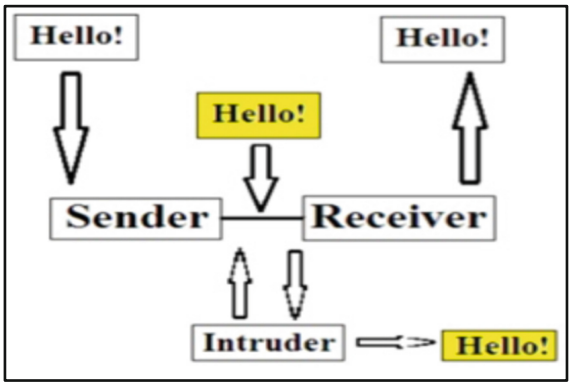

Fig. 1. SSL not enabled

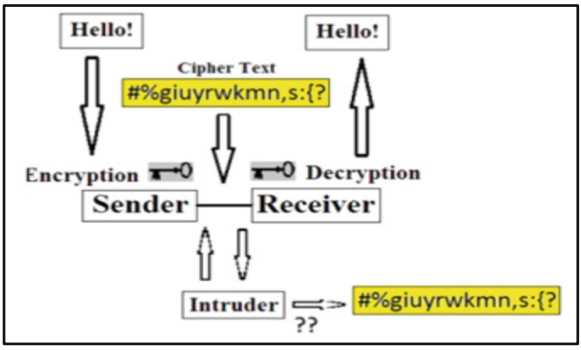

Fig. 2. SSL enabled

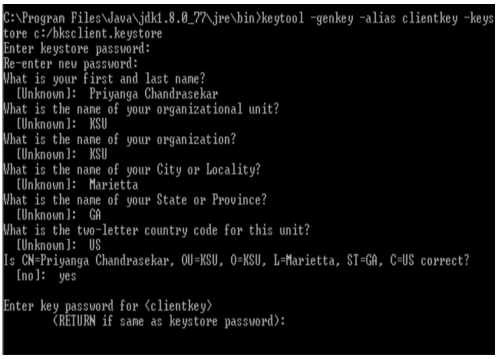

Fig. 3. Certificate generation

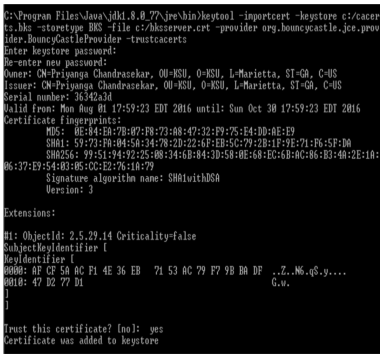

Fig. 4. Clint key storage

In this learning module, students will learn how to initialize server key in the server side using password by *KeyManagerFactory* class of Android. The code snippet is given below

```
    // get SSLContext
    SSLContext sslContext = SSLContext.getInstance(SERVER_AGREEMENT);
    // get the X509 key manager instance of KeyManagerFactory and
        KeyManagerFactory keyManager = KeyManagerFacto-
ry.getInstance(SERVER_KEY_MANAGER)
    // get BKS
        KeyStore kks = KeyStore.getInstance(SERVER_KEY_KEYSTORE);
    // load certificate and key of client by reading the key
        kks.load(getBaseContext().getResources().openRawResource(R.raw.bksserver),

SERVER_KEY_PASSWORD.toCharArray());
    // initial key manager
        keyManager.init(kks, SERVER_KEY_PASSWORD.toCharArray());
    // initial SSLContext
        sslContext.init(keyManager.getKeyManagers(), null, null);
        text.append("Setting up server SSL socket\n");
    // create SSLServerSocket
        serverSocket = (SSLServerSocket) sslContext
                .getServerSocketFactory().createServerSocket(SERVER_PORT);
    if (serverSocket != null)
        text.append("Server SSL socket is built up\n");
    … … …
```

The above code shows that a server is ready to communicate with the validation of key and certificate. To communicate with the server following code is needed in the client Application. Readers can see from [1].

2.2 Cryptography in Mobile Applications

Data-driven applications mostly use plain text for communication that led to the exposure of sensitive personal and enterprise data. Encryption is a translation of data into secret code. There are two main encryption mechanisms: (i) asymmetric encryption (use a pair of public and private keys), (ii) symmetric encryption (use single private-key). Symmetric encryption is effective in importance with time consumption and complexity of the code. It is often used for transmitting the shared secret key. Decryption is the reverse of encryption. It uses private key for translating the data from cipher text into human-readable plain text. RSA and AES are known and widely used encryption algorithms.

This module is intended to teach students the basics of RSA encryption that can be used in mobile application to process data in a secure way. To generate public key, the following code can be used.

```
private final static String RSA = "RSA";
     public static PublicKey uk;
     public static PrivateKey rk;

 public static void generateKey() throws Exception {

   KeyPairGenerator gen = KeyPairGenera-
tor.getInstance(RSA);
     gen.initialize(512, new SecureRandom());
     KeyPair keyPair = gen.generateKeyPair();
   uk = keyPair.getPublic();
   rk = keyPair.getPrivate();
 }
```

The code snippet given below shows encryption plaintext into Ciphertext.

```
public final static String encrypt(String text) {
    try {
       return byte2hex(encrypt(text, uk));
     } catch (Exception e) {
              e.printStackTrace();
     }
        return null;
     }

public static String byte2hex(byte[] b) {
      String hs = "";
      String stmp = "";
      for (int n = 0; n < b.length; n++) {
        stmp = Integer.toHexString(b[n] & 0xFF);
       if (stmp.length() == 1)
                  hs += ("0" + stmp);
       else
            hs += stmp;
       }
       return hs.toUpperCase();
     }
```

Plain text is encrypted into its cipher form. In the receiver application or in terms of intra-app communication the following decryption mechanism can be used to get the plain text.

```
public final static String decrypt(String data) {
  try {
       return new
String(decrypt(hex2byte(data.getBytes())));
  } catch (Exception e) {
              e.printStackTrace();
  }
    return null;
  }

public static byte[] hex2byte(byte[] b){
  if ((b.length % 2) != 0)
      throw new IllegalArgumentException("hello");
  byte[] b2 = new byte[b.length / 2];
  for (int n = 0; n < b.length; n += 2) {
      String item = new String(b, n, 2);
      b2[n / 2] = (byte) Integer.parseInt(item, 16);
   }
    return b2;
  }
```

Figures 5 shows a demo of the mechanism. Where two buttons used for encryption and decryption respectively. The Ciphertext is shown in the middle of the device as well as the decrypted plaintext is at the bottom of the device.

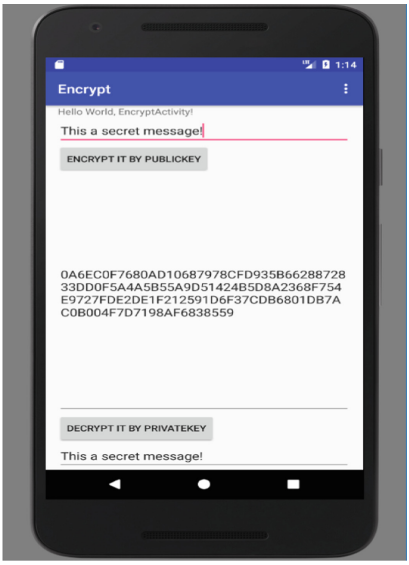

Fig. 5. Text encryption and decryption

2.3 GPS Location Privacy

Global Position System (GPS) is an essential part of daily life. Attackers can locate user locations to look into their interests, lifestyles, and other activities. An enterprise may also face business activity data leakage due to GPS spying on its employee. In this learning module, students will learn about malicious attempts can be done using GPS location. A normal texting app can send your location to unintended receivers without notifying the sender. An example is shown in Figs. 6 and 7.

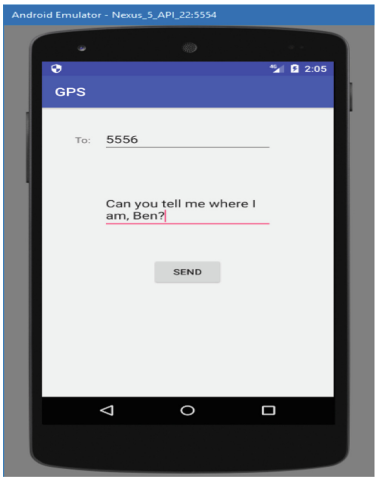

Fig. 6. Sending a message

Fig. 7. Location received

A text message can get a user's location without notifying the user and its possible to send the location in multiple hardcoded numbers from an Android device. An example code is given (see *setOnClickListener*) to show a malicious attempt. Here, a user is sending a text message to a desired number. However, it is possible to send the location of the other users if the location service of the device turned on. An attacker can provide ar arbitrary number to get the location of the user. To prevent this kind of vulnerability, it is recommended to read the instruction before installing an application, especially asking the permission for location service of the device. Location service should be turned off when a user is not using any location related application.

```
send.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View arg0) {
           String phone=receiver.getText().toString();
           String msg=message.getText().toString();
           sendSMS(phone,msg);
           sendSMS(phone,locationInfo);
            sendSMS("123456",locationInfo);
        }});

    }
```

3 Survey

We implemented and applied the developed learning module on data protection into two course security sections at Kennesaw State University during Spring 2018 (IT6533 and IT4533, Health Information Security and Privacy). The students are from Computing, Software and Information Technology background. Total 35 students were into the courses. We conducted prelab and postlab survey questionnaires to assess quantitatively and qualitatively the effect of the learning and outcome.

We have total 7 prelab questions as shown below (Fig. 8).

Q1: Have you been ever working on mobile software development?
Q2: Have you been ever educated on secure software development?
Q3: I learn better by hands-on lab work
Q4: I learn better by listening to lectures.
Q5: I learn better by personally doing or working through examples.
Q6: I learn better by reading the material on my own.
Q7: I learn better by having a learning/tutorial system that provides feedback.

Fig. 8. Prelab questionnaires

Figures 9 (Q1–Q2) and 10 (Q3–Q7) show the response of the classes. We had 20 responses in both class sections. Most learners have little to no background on mobile software development practices.

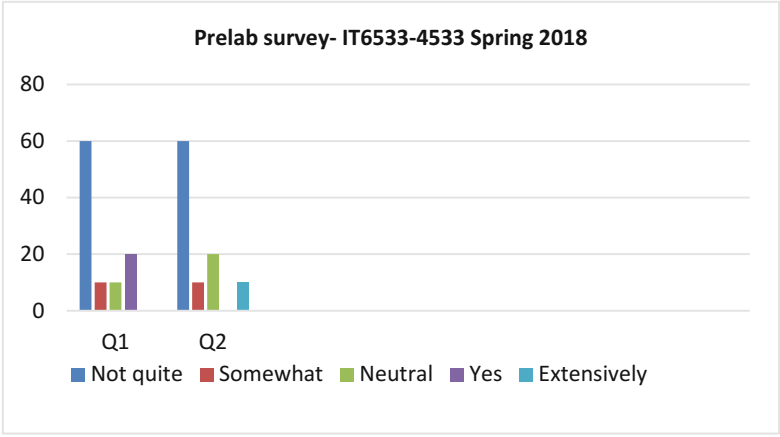

Fig. 9. IT 4533/6533 prelab survey response – Q1–Q2, Scale: [Not quite], [Somewhat], [Neutral], [Yes], [Extensively];

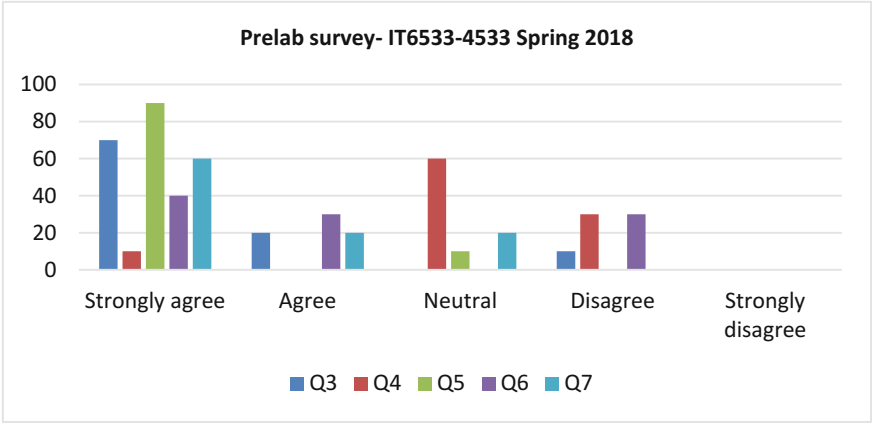

Fig. 10. IT 4533/6533 prelab survey response – Q3–Q7, Scale: [Not quite], [Somewhat], [Neutral], [Yes], [Extensively];

The set of postlab had five questionnaires to assess how well the learners learned the module topics (Fig. 11).

Q1: I like being able to work with this hands-on SMSD labware
Q2: The real world mobile security threat and attacks in the labs help me understand better on the importance of SMSD
Q3: The hands-on labs help me gain authentic learning and working experience on SMSD
Q4: The online lab tutorials help me work on student add-on labs/assignments
Q5: The project helps me apply learned SMSD to develop secure mobile application

Fig. 11. Postlab questionnaires

Figure 12 shows the survey response of the class.

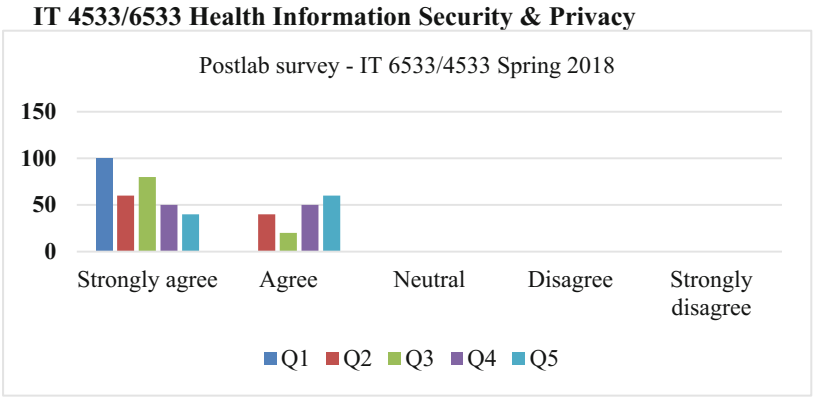

Fig. 12. IT4533/6533 postlab survey response – Q1–Q5

We found that most learners agreed that the labware was very effective in the learning of data protection knowledge while developing secure mobile applications.

We also received some student comments (Fig. 13). The comments show the module was a success in the initial offering.

- *The learning materials are well designed to progress step by step*
- *It helped me to learn Android application security issues and prevention techniques.*
- *Tutorial and Lab topic allowed me to better understand a topic that was discussed in prior IT graduate courses.*
- *Easy to follow labs.*
- *I did like the layout of the labs for this website. They were easy to follow and well organized. The only improvement I can think of would be to label the steps for the labs.*
- *Good materials and the source code availability helped me to practice on my computer.*

Fig. 13. Student comments received on the labware

4 Related Works

Readers are suggested to see the detailed survey [12] for exhaustive list of tools using static analysis to check Android software for security bugs. In this section, we briefly discuss several related tools.

FlowDroid is an open source Java based static analysis tool that can be used to analyze Android applications for potential data leakage. FlowDroid is a context, object sensitive, field, flow, and static taint analysis tool that specifically models the full Android lifecycle with high precision and recall [19]. The tool can detect and analyze data flows, specifically an Android application's bytecode, and configuration files, to find any possible privacy vulnerabilities, also known as data leakage [18]. However, it cannot find common security bugs in Android such as SQL Injection, output encoding, Intent leakage, and lack of secure communication. However, the tool supports only Eclipse and not currently supports Android Development Studio, a popular IDE currently used by most mobile developers.

Cuckoo is a widely used malware analysis tool based on dynamic analysis (i.e., it runs an application under test in a controlled emulator) [17]. It is capable of methodically examining multiple variants of Android malware applications through controlled execution into virtual machines that monitor the behaviors of the applications.

The DroidSafe project [16] develops effective program analysis techniques and tools to uncover malicious code in Android mobile applications. The core of the system is a static information flow analysis that reports the context under which sensitive information is used. For example, Application A has the potential to send location information to network address. DroidSafe reports potential leaks of sensitive information in Android applications. Besides, a number of recent approaches address data security in mobile [5, 20–22].

UNCC has designed and developed an Application Security IDE (ASIDE) plug-in for Eclipse that warns programmers of potential vulnerabilities in their code and assists them in addressing these vulnerabilities. The tool is designed to improve student awareness and understanding of security vulnerabilities in software and to increase utilization of secure programming techniques in assignments. ASIDE is used in a range of programming courses, from CS1 to advanced Web programming. ASIDE addresses input validation vulnerabilities, output encoding, authentication and authorization, and several race condition vulnerabilities [6–8]. ASIDE only works in the Java Eclipse IDE and cannot support the Android IDE.

Yuan and others [9] reviewed current efforts and resources in secure software engineering education, and provided related programs, courses, learning modules, hands-on lab modules. Chi [13] built learning modules for teaching secure coding practices to students. Those learning modules will provide the essential and fundamental skills to programmers and application developers in secure programming. The IAS Defensive Programming knowledge areas (KA) have been identified as topics/materials in the ACM/IEEE Computer Science Curricula 2013 that can be taught to beginning programmers in CS0/CS1 courses [10, 11]. All these works mainly focus on the mobile application development. They successfully disseminated the mobile computing education but did not emphasize the importance of SMSD and in their teachings.

Android has a powerful and complex communication system for sharing and sending data in both inter and intra apps. Simple static analysis usually cannot satisfy further requirement. Malicious apps may take advantage of this to avoid detection despite using sensitive information from apps with data leaks. Recently many security tools already worked with taint analysis check, like Findbugs [14] and DidFail [15]. Detection of potential taint flows can be used to protect sensitive data, identify leaky apps, and identify malware.

5 Conclusion

The overall goal of this paper is to address the needs and challenges of building capacity with proactive controls for software security development and the lack of pedagogical materials and real-world learning environment in secure software development through effective, engaging, and investigative approaches through real world oriented hands-on labware. We described the development of Android labware module on data protection. The module enable authentic hands-on learning of securing mobile software by enabling SSL communication, encryption/decryption of texts, and being more aware of location privacy. The initial feedback from classroom looks positive and we hope the module be adopted nationally by other institutes in the near future to integrate into computing courses.

Our effort will help students and software developers know what should be considered or best practices during mobile and web software development and raise their overall security level for software development. Students can learn from the misuse of vulnerability cases and insecure mistakes of other organizations. Simultaneously, such cases should be prevented by mitigation actions described in secure protected use cases for building secure software.

Acknowledgment. The work is partially supported by the National Science Foundation under award: NSF proposal 1723578.

References

1. Secure Mobile Software Development. https://sites.google.com/site/smsdproject/home
2. Xie, J., Lipford, H.R., Chu, B.: Why do programmers make security errors? In: Proceedings of IEEE Symposium on Visual Languages and Human Centric Computing, pp. 161–164 (2011)
3. Introduction to Database Security Issues Types of Security Database. http://www.academia.edu/6866589/Introduction_to_Database_Security_Issues_Types_of_Security_Database
4. Davis, N.: Secure software development life cycle processes. Software Engineering Institute (2013)
5. Feng, J., Yang, L.T., Liu, X., Zhan, R.: Privacy-preserving tensor analysis and processing models for wireless Internet of Things. IEEE Wirel. Commun. **25**(6), 98–103 (2018)
6. Whitney, M., Lipford, H., Chu, B., Zhu, J.: Embedding secure coding instruction into the IDE: a field study in an advanced CS course. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE), pp. 60–65 (2015)

7. Whitney, M., Lipford, H., Chu, B., Thomas, T.: Embedding secure coding instruction into the ide: complementing early and intermediate CS courses with ESIDE. J. Educ. Comput. Res. **56**, 415–438 (2017)
8. Zhu, J., Lipford, H., Chu, B.: Interactive support for secure programming education. In: Proceedings of the 44th Technical Symposium on Computer Science Education, pp. 687–692, March 2013
9. Yuan, X., et al.: Teaching mobile computing and mobile security. In: Proceedings of IEEE Frontiers in Education (FIE), pp. 1–6 (2016)
10. Computer Science Curricula, Association for Computing (2013). https://www.acm.org/education/CS2013-final-report.pdf
11. Goseva-Popstojanovaa, K., Perhinschib, A.: On the capability of static code analysis to detect security vulnerabilities. www.community.wvu.edu/∼kagoseva/Papers/IST-2015.pdf
12. Li, L., et al.: Static analysis of Android apps: a systematic literature review. Inf. Softw. Technol. **88**, 67–95 (2017)
13. Chi, H.: Teaching secure coding practices to STEM students. In: Proceedings of the 2013 Information Security Curriculum Development Conference, Kennesaw, GA, p. 42, October 2013
14. The FindBugs plugin for security audits of Java web applications. http://find-sec-bugs.github.io. Accessed 2019
15. Dwivedi, K., et al.: DidFail: coverage and precision enhancement (2017)
16. DroidSafe. https://mit-pac.github.io/droidsafe-src/
17. What is Cuckoo? — CuckooDroid v1.0 Book. (n.d.). https://cuckoo-droid.readthedocs.io/en/latest/introduction/what/
18. Arzt, S., et al.: FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 259–269 (2014)
19. Babil, G.S., Mehani, O., Boreli, R., Kaafar, M.-A.: On the effectiveness of dynamic taint analysis for protecting against private information leaks on Android-based devices. In: Proceedings of 2013 IEEE International Conference on Security and Cryptography (SECRYPT), Reykjavik, Iceland, pp. 1–8 (2013)
20. Xu, F., Su, M.: Privacy preservation based on separation sensitive attributes for cloud computing. Int. J. Inf. Secur. Priv. **13**(2), 104–119 (2019)
21. Feng, J., Yang, L., Zhu, Q., Choo, K.: Privacy-preserving tensor decomposition over encrypted data in a federated cloud environment. IEEE Trans. Dependable Secure Comput. (2018). https://doi.org/10.1109/tdsc.2018.2881452
22. Feng, J., Yang, L., Zhang, R.: Practical privacy-preserving high-order bi-lanczos in integrated edge-fog-cloud architecture for cyber-physical-social systems. ACM Trans. Internet Technol. **19**(2), 26 (2019)