

Oct 12th, 1:25 PM - 1:50 PM

Automatic Security Bug Detection with FindSecurityBugs Plugin

Hossain Shahriar

Kennesaw State University, hshahria@kennesaw.edu

Kmarul Riad

Kennesaw State University, aislamri@students.kennesaw.edu

Arabin Talukder

KSU, mtalukd1@students.kennesaw.edu

Hao Zhang

KSU, hzhang13@students.kennesaw.edu

Zhuolin Li

Kennesaw State University, zli29@students.kennesaw.edu

Follow this and additional works at: <https://digitalcommons.kennesaw.edu/ccerp>



Part of the [Information Security Commons](#), [Management Information Systems Commons](#), and the [Technology and Innovation Commons](#)

Shahriar, Hossain; Riad, Kmarul; Talukder, Arabin; Zhang, Hao; and Li, Zhuolin, "Automatic Security Bug Detection with FindSecurityBugs Plugin" (2019). *KSU Proceedings on Cybersecurity Education, Research and Practice*. 6.
<https://digitalcommons.kennesaw.edu/ccerp/2019/research/6>

This Event is brought to you for free and open access by the Conferences, Workshops, and Lectures at DigitalCommons@Kennesaw State University. It has been accepted for inclusion in KSU Proceedings on Cybersecurity Education, Research and Practice by an authorized administrator of DigitalCommons@Kennesaw State University. For more information, please contact digitalcommons@kennesaw.edu.

Abstract

The security threats to mobile application are growing explosively. Mobile app flaws and security defects could open doors for hackers to easily attack mobile apps. Secure software development must be addressed earlier in the development lifecycle rather than fixing the security holes after attacking. Early eliminating against possible security vulnerability will help us increase the security of software and mitigate the consequence of damages of data loss caused by potential malicious attacking. In this paper, we present a static security analysis approach with open source FindSecurityBugs plugin for Android Studio IDE. We demonstrate that integration of the plugin enables developers secure mobile application and mitigating security risks during implementation time.

The security threats to mobile application are growing explosively. Mobile app flaws and security defects could open doors for hackers to easily attack mobile apps. Secure software development must be addressed earlier in the development lifecycle rather than fixing the security holes after attacking. Early eliminating against possible security vulnerability will help us increase the security of software and mitigate the consequence of damages of data loss caused by potential malicious attacking. In this paper, we present a static security analysis approach with open source FindSecurityBugs plugin for Android Studio IDE. We demonstrate that integration of the plugin enables developers secure mobile application and mitigating security risks during implementation time.

The security threats to mobile application are growing explosively. Mobile app flaws and security defects could open doors for hackers to easily attack mobile apps. Secure software development must be addressed earlier in the development lifecycle rather than fixing the security holes after attacking. Early eliminating against possible security vulnerability will help us increase the security of software and mitigate the consequence of damages of data loss caused by potential malicious attacking. In this paper, we present a static security analysis approach with open source FindSecurityBugs plugin for Android Studio IDE. We demonstrate that integration of the plugin enables developers secure mobile application and mitigating security risks during implementation time.

Location

KSU Center Rm 400

Disciplines

Information Security | Management Information Systems | Technology and Innovation

INTRODUCTION

With the increased demands of mobile applications in recent years, we have also witnessed numerous major cyber-attacks, resulting in stolen personal credit card numbers, leakage of classified information vital for national defense, industrial espionage resulting in major financial losses, and many more consequences. Hackers have managed to make secure computing a more difficult task. This has resulted in the need for increased secure mobile software development tools and resources for professionals (Shahriar et al., 2018; Qian et al., 2017).

Most mobile security vulnerabilities should be addressed and fixed in the software development phase. If all the mobile applications are secure or have less security flaws and vulnerabilities, the security threat risks will be greatly reduced. Computer users, managers, and developers agree that we need software and systems that are "more secure". Such efforts require support from both the education and training communities to improve software assurance, particularly in writing secure code.

Many open source static Java code analysis tools help developers to maintain and clean up the code through the analysis performed without actually executing the code such as Eclipse IDE (2019), IntelliJ IDE (2019), and FindBugs (2019). These tools focus on finding probable bugs such as inconsistencies, helping improve the code structure, conform source code to guidelines, and provide quick fixes. Source code analysis tools, also referred to as Static Application Security Testing (SAST) Tools, are designed to analyze source code and to help to find security flaws with a high confidence that what's found is indeed a flaw (readers can see the survey (Li et al., 2017) for list of exhaustive state-of-the-art tools). However, there is no tool that can assist developers securing applications within the development environment.

There are many open source static Java code analysis tool that helps developers to maintain and clean up the code through the analysis performed without actually executing the code such as Eclipse IDE, IntelliJ IDE, FindBugs Plugin. These tools focus on finding probable bugs such as inconsistencies, helping improve the code structure, conform your code to guidelines, and provide quick-fix. In general, SCA tools are used to ensure code quality from the very beginning and to make software development more productive. The security vulnerability checking is not their major task. Source code analysis tools, also referred to as Static Application Security Testing (SAST) Tools, are designed to analyze source code and to help to find security flaws with a high confidence that what's found is indeed a flaw.

FindSecurityBugs (2019) is a static code analysis plugin for the FindBugs (a plugin for Eclipse). It specializes in finding security issues in Java code by searching for security. It can be used to scan Java applications. However, there is little or no effort to integrate with Android Studio Development IDE to assist developer building mobile applications securely. Integrating tools within IDE not only opens the door for designing custom flaw detectors but also to report the emerging security threats during the software development phase with immediate feedback to the developer on rather than finding vulnerabilities much later in the development cycle.

In this paper, we introduce FindSecurityBugs (FSB) plugins, which we ported to Android Development Studio to mitigate security issues during application development time. FSB allows developer to design custom security vulnerability detectors. We have designed and developed a number of new security flaw detectors with FindSecurityBugs plug-in for Android mobile software development based on current OWASP 2017 top 10 mobile risks to increase the security vulnerability checking coverage.

This paper is ^{KSU Proceedings on Cybersecurity Education, Research and Practice, Event 6 [2019]} organized as follows. Section II discusses related work. Section III is an overview of tainted data flow analysis, followed by examples of integrated FSB plugins for two common mobile software security bugs – intent spoofing and SQL injection. Section IV concludes the paper.

RELATED WORKS

Readers are suggested to see the detailed survey [18] 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 (Babil et al., 2013). 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 (Artz et al., 2013). 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 (2019) is a widely used malware analysis tool based on dynamic analysis (i.e., it runs an application under test in a controlled emulator). 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 [20] 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.

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 (Whitney et al., 2005). ASIDE only works in the Java Eclipse IDE and cannot support the Android IDE.

Yuan et al. (2016) reviewed current efforts and resources in secure software engineering education, and provided related programs, courses, learning modules, hands-on lab modules. Chi (2013) 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 (CS Curricula, 2013). 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 requirements. 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 (2019) and Dwivedi et al. (2017). Detection of potential taint flows can be used to protect sensitive data, identify leaky apps, and identify malware.

VULNERABILITY DETECTORS WITH FINDSECURITYBUGS

Taint analysis Detectors for Secure Mobile Software

To meet the ever-increasing demand for high quality information security professionals with expertise in SMSD, we built innovative Android vulnerability detectors with an open source FindSecurityBugs API plugin for the popular Android Studio IDE based on the most current OWASP 2017 top 10 mobile security risks (OWASP, 2017) in the category of SQL injection, unintended data leakage, insecure communication, insecure data storage vulnerability detectors. For example, the built in detectors can recognize a vulnerability of SQL injection and data leakage in an Android mobile application program, which may face the threat of potential malicious code injection, and then issue a warning on the code line. Following the provided options, students or developers can enforce a new secure statement to replace the unsecure statement.

Talukder et al. (2019) developed a plug-in for Android Studio IDE that can parse Android java source code, identify specific API calls, warn the potential vulnerabilities, recommend security solutions, and replace code statements. The plugin tool can recognize a vulnerability of SQL injection and data leakage in an Android mobile application program, which may face the threat of potential malicious code injection, and then issue a warning on the code line. Once the developer clicks the warning icon, secure coding prevention and protection options are shown in the Android Studio.

For many Android security vulnerabilities and flaws on the top 10 mobile risks by OWASP (2017) and other new identified unlisted flaws we need to develop our own customized detectors. Figure 1, show the conception of taint analysis. Here a source is the resource from which data is read, may either come from outside or internal. A sink is a resource to which data is written or sent. Also, a piece of data is tainted if it originates from a sensitive source.

Based on OWASP top 10 mobile risk reports, we have developed more vulnerability detectors with FindsecurityBugs for secure Android software development in the categories of Unintended Data Leakage, Intent Interception and Spoofing, Content Provider Data Sharing, Input Validation and Output Encoding. Section A shows a SQL injection detector and section B shows unintended data leakage detector for clipboard cache memory data leakage.

Taint-based Broadcast Intent Detector

Broadcast receivers are used to handle asynchronous requests initiated via an intent. By default, receivers are exported and can be invoked by any other application. If your BroadcastReceivers is intended for use by other applications, you may want to apply security permissions to receivers using the <receiver> element within the application manifest. This will prevent applications without appropriate permissions from sending an intent to the BroadcastReceivers.

There two types of intent: Explicit intents has its explicit recipient and Implicit intents does not name its explicit recipient, and it will notify an appropriate component based on the specification of the intent.

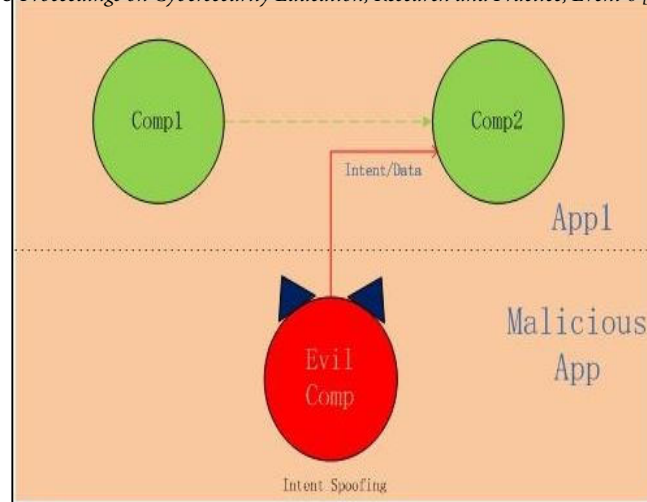


Figure1: Intent spoofing

Figure 1 illustrates the concept of intent spoofing where comp1 and comp2 are two Android components (Activity, Service, or BroadCastReceiver) and app1 is a victim. The comp2 in App1 expects to get intent with data from Comp1 in the same App1 but instead, it gets a malicious injection via an implicit intent sent by a Malicious app. This is an inter-app intent spoofing which can be prevented by explicit intent, setting an exported attribute to false, claiming permission requirement by app1

To receive an implicit intent an Android component must register the implicit intent with an intent filter specifying the kinds of intents it is interesting. Implicit intents are useful for an app to request a service function without knowing exactly the service function provider (Talukder, Shahriar & Haddad, 2019). It provides flexibility in run-time binding of components.

1) Intent spoofing is an attack where a malicious application induces or injects undesired behavior to a component via implicit intent, which only expects to receive intents from other components within the same app. By default, a component only receives intents from other components in the same application, but it can also accept intents from other apps if the android: exported attribute is set in the manifest XML. Figure 2 (a) shows an explicit broadcast intent is sent where FindSecurityBugs caught it as a vulnerable intent.

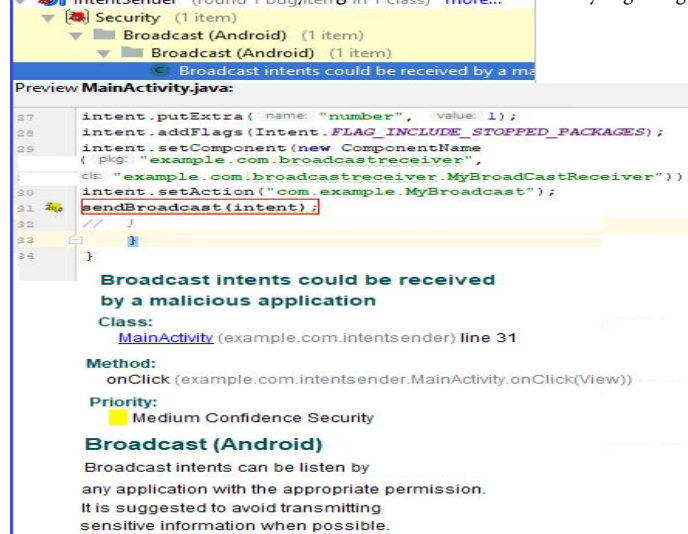


Figure 2: Vulnerable Intent found by FindSecurityBugs.

The solution of explicit broadcast intent security issue could be adding the following code fragment.

Solution (if possible):

```
Intent i = new Intent();
i.setAction("com.secure.action.UserConnected");

sendBroadcast(v1);
```

Figure 2(b): Solution of broadcast Intent security issue.

2) To establish communication between two applications we can use broadcast intent. However, in this case we may need to add an additional component such as explicit permission to build a secure communication between two applications. As we know broadcast receiver receives intent from other application by parsing intent filters action name. FindSecurityBugs plugin again detects this type of communication as a vulnerable coding practice. Figure 3 shows FindSecurityBugs plugin detects the intent as a bug.



Figure 3: Broadcast intent detected by FindSecurityBugs

To secure communication between two applications, we can use customized permission in the sender applications manifest. See the code fragment in Figure 4.

```
<manifest ...>

  <!-- Permission declaration -->
  <permission android:name="my.app.PERMISSION" />

  <receiver
    android:name="my.app.BroadcastReceiver"
    android:permission="my.app.PERMISSION"> <!-- Permission enforcement -->
    <intent-filter>
      <action android:name="com.secure.action.UserConnected" />
    </intent-filter>
  </receiver>

  ...
</manifest>
```

Figure 4: Permission to secure intent

This permission can be used in the receiver application's manifest to detect the intent that is sent from the sender. In this case, other application will not detect the intent because of the unavailability of the permission in their manifest. Broadcast class have an additional method parameter to its "sendBroadcast" method that is "receiverspermission". To send an intent with customized permission we can use this method. See an code fragment below:

```
Intent intent = new Intent("com.example.CUSTOM_ACTION");
intent.putExtras(bundle);
sendBroadcast(intent,"permission");
```

Now only those application will receive this intent who has the same permission in their manifest

Taint-based SQL Injection Detector

SQL injection is a code insertion technique used to attack data driven applications, in which malicious code is inserted to normal SQL statement to dump contents from database. SQL injection exploits security vulnerabilities of application, for example, taking use of a user input to embed malicious code to a hard code SQL statement.

There are several forms of SQL injection, consisting of direct insertion code to user input variables and then concatenated with SQL statements to be executed or other less direct code insertion technique. Some of them are listed as below:

1) Incorrectly filtered escape characters: This form occurs if user input is passed to SQL statement without filtering escape characters. Implementation is illustrated in the following statement.

```
"SELECT * FROM users WHERE username= "+username+"'"
```

where the variable username can be crafted by attackers, either by inputting an always true clause or by commenting out the rest of query statement. What's more, by inserting a semicolon, attackers are able to execute separate SQL statement in this case.

2) Incorrect type handling: This form of injection takes place when no appropriate type checking is performed. The implementation of this form can be the same as the previous one. There are still many forms to perform injection, the point that an injection works is by prematurely terminating a text string and appending a new command.

3) Defense strategy for SQL injection: In addition to input validation to eliminate the malicious injection we also should use secure parameterized query statements with placeholders to receive parameters instead of embedding user input to query statement. Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied. This strategy will also solve the problem if there is not a type handling mechanism, because a placeholder can only receive value of the given type.

Vulnerable code fragment:

```
cursor = db.rawQuery("SELECT * FROM usertable WHERE _id='" +
info + "'", null);
```

Secure parameterized query:

```
String s1 = input.getText().toString();
cursor = db.rawQuery("SELECT * FROM
usertable WHERE _id= ? ", s1);
```

Here is a sample vulnerable SQL injection code detected by the SQL injection detector after scanning an Android app with the following snippet

```
EditText input;
String info = input.getText().toString()
```

KSU Proceedings on Cybersecurity Education, Research and Practice, Event 6 [2019]

```
cursor = db.rawQuery("SELECT * FROM usertable WHERE _id='" +
info + "'", null);
```

Figure 5 shows a warning alert and a solution hint in the Android Studio IDE after detector finds the SQL injection vulnerability:

Vulnerability alert! Use parameterized query with placeholders ("?") to receive input parameters instead of embedding user input to query statement with string concatenation ("+") to avoid malicious SQL injection.

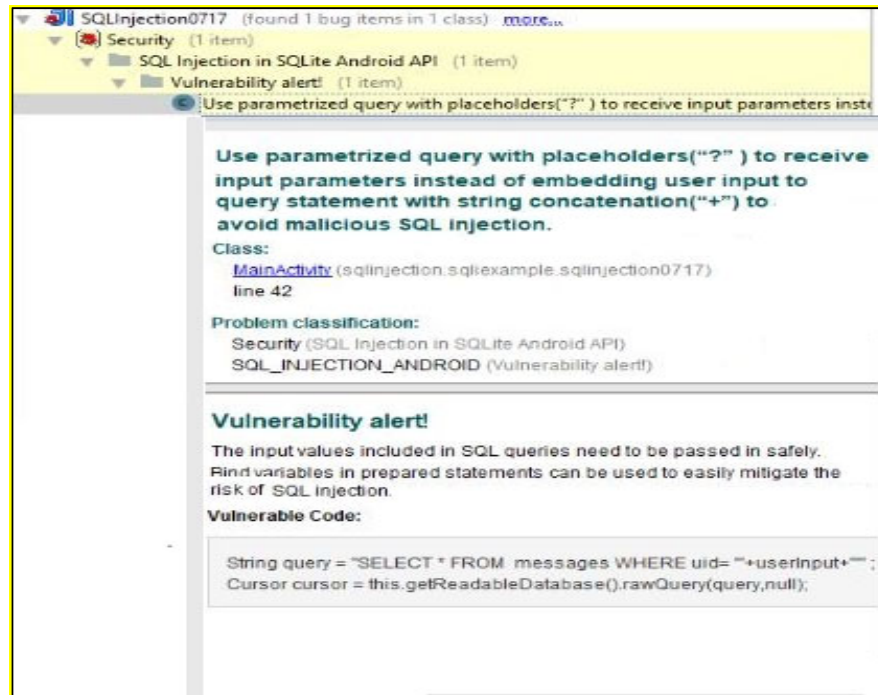


Figure 5: Detector for Android SQL injection

CONCLUSION AND FUTURE WORK

In this paper, we introduced a taint analysis-based Android plugin that can be used to develop secure mobile software while addressing security concerns commonly found in mobile applications. Our focus is aimed to reduce OWASP Top 10 Mobile Security flaws before applications are deployed. This effort not only will overcome the shortage of tools within development environment, but also resources for professionals towards secure mobile software development. We are currently working to expand the capability of the plugin tools by developing detectors for common (e.g., insecure communication between mobile applications) and advanced security concerns (e.g., permission escalation, content sniffing).

ACKNOWLEDGEMENT

The work is partially supported by National Science Foundation (Award #1723578); KSU Office of Vice President Research (OVPR 2018-2019) Award, and USG ALG Grants (M54, M87, #422, #429).

REFERENCES

- Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Mcdaniel, P. (2013). FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps, *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 259-269.
- Babil, G., Mehani, O., Roksana, B., Kaafar, M. (2013). On the effectiveness of dynamic taint analysis for protecting against private information leaks on Android-based devices. *Proc. of the 2013 International Conference on Security and Cryptography (SECRYPT)*, Reykjavik, Iceland, pp. 1-8.
- Chi, H. (2013). Teaching Secure Coding Practices to STEM Students, *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference*, Kennesaw, GA, USA.
- Cuckoo. (2019). What is Cuckoo? — CuckooDroid v1.0 Book. (n.d.). Retrieved from <https://cuckoo-droid.readthedocs.io/en/latest/introduction/what/>
- CS Curricula. (2013). Association for Computing, <https://www.acm.org/education/CS2013-final-report.pdf>
- DroidSafe. (2019). <https://mit-pac.github.io/droidsafe-src/>
- Dwivedi, K., Yin, H., Bagree, P., Tang, X., Flynn, L., Klieber, W., Snaveley, W. (2017). DidFail: Coverage and Precision Enhancement, Technical Report, Carnegie Mellon University, <https://apps.dtic.mil/docs/citations/AD1044883>
- Eclipse IDE. (2019). <https://www.eclipse.org/ide/>
- FindBugs. (2019). <http://findbugs.sourceforge.net/>
- FindSecurityBugs. (2019). <https://find-sec-bugs.github.io/>
- IntelliJ IDEA. (2019). <https://www.jetbrains.com/idea/>
- FindBugs. (2019). Plugin for security audits of Java web applications. <http://find-sec-bugs.github.io>.
- Li, L., Bissyand'e, T., Papadakis, M., Rasthofer, S., Bartela, A., Octeau, D., Kleina, J., Traona, Y. (2017). Static Analysis of Android Apps: A Systematic Literature Review, *Information and Software Technology*, Volume 88, pp. 67-95.
- OWASP. (2017). Mobile Security Project - Top Ten Mobile Risks, https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks
- Qian, K., Shahriar, H., Wu, F., Tao, L., Bhattacharya, P. (2017). Labware for Secure Mobile Software Development (SMSD) Education, *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 375-375.
- Shahriar, H., Qian, K., Talukder, M., Patel, N., and Lo, D. (2018). Mobile Software Security Risk Assessment with Program Analysis. *Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, Taipei, Taiwan, 2 pp.
- Talukder, Md., Shahriar, S., Qian, K., Rahman, M., Ahmed, S., Agu, E. (2019). DroidPatrol: A Static Analysis Plugin For Secure Mobile Software Development, *Proc. of 43rd IEEE Annual Computer Software and Applications Conference (COMPSAC)*, pp. 565-569.
- Talukder, Md., Shahriar, S., and Haddad, H. (2019), Point-of-Sale Device Attacks and Mitigation Approaches for Cyber-Physical Systems, *Cybersecurity and privacy in Cyber Physical Systems*, CRC Press, pp. 368-383.
- Whitney, M., Lipford, H., Chu, B., and Zhu, J. (2015). Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course, *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 60-65.
- Yuan, X., Williams, K., McCrickard, D., Hardnett, C., Lineberry, L., Bryant, K., Xu, J., Esterline, A., Liu, A., Mohanarajah, S., Rutledge, R. (2016). Teaching mobile computing and mobile security, *Proc. of Frontiers in Education (FIE)*, 2016, pp. 1-6.