Safeguarding Mobile Applications With Secure Development Life Cycle Approach

In today's age of bring your own device (BYOD), the smartphone is one of the preferred mobile devices to access enterprise information. Software is a key component in any information technology asset. Smart devices either are embedded with application software or allow users to install software on the devices to add functions that accomplish intended objectives. Hence, applications are vital to mobile devices. Securing these applications from security vulnerabilities and risk is fundamental.[1]

This article focuses on secure development practices in mobile applications development and suggests a few open-source security tools to perform an application security assessment to strengthen mobile applications.

Web Application Security Issues Lead to Enterprise Breaches

Major information security breaches have occurred in the last few years. Security researchers took a close look at the underlying reasons for some of these breaches, and their studies reveal that the security of a web application is of paramount importance to the enterprise perimeter and gateway-level security.[2, 3, 4, 5, 6, 7, 8]

An insecure web application can compromise the best enterprise security arrangements and can help adversaries steal data and gain a foothold into the enterprise's internal network.

Mobile Application Security Issues

Security issues are no different in the case of mobile applications—wherein the application is downloaded from the Internet (e.g., Apple Store or Google Play Store) and installed in the user's device. Similar to web applications that are exposed on the Internet, mobile applications that are installed in BYOD devices are entry points to the enterprise network.

Installed mobile applications, if not protected appropriately, can be reverse engineered to get their source code, which is in human-readable form. Platforms such as iOS and Android—the two most popular mobile platforms today—are not immune to the threat of reverse engineering. A few easy steps and widely available (often free) tools make it easy for an attacker to:

- Extract the installed application from the mobile device
- Analyze or reverse engineer the code to find vital information, e.g., business logic, the application programming interface (API) used and embedded internal URLs
- Modify the code to change application behavior
- Inject malicious code

Do you have something to say about this article?
Visit the *Journal* pages of the ISACA® website *(www.isaca.org/journal)*, find the article and click on the Comments link to share your thoughts.

Sakthivel Rajendran, CISA, CRISC, CISM, CEH, GMOB
Is an information security manager in India and works with a major global health care company. He has more than a decade of experience in IT security. His area of focus is security of emerging technologies. He can be reached at sakthiindian@gmail.com.

Reverse engineering mobile applications is a security concern that enterprises should consider. Code obfuscation is a well-known technique that makes reverse engineering of a mobile application difficult,[9] but this technique is often ignored by the development community. Mobile applications exhibit the following security weaknesses:[10]

- Lack of privacy considerations
- Lack of binary protection
- Insecure data storage
- Transport security
- Weak server-side controls

Research by two security experts representing different security firms reveals that the mobile banking applications of top influential banks around the world have many common security vulnerabilities.[11, 12] These researchers performed their tests on the mobile application (client side) and excluded any server-side testing (back end). The client side represents only a small portion of the attack surface of mobile banking, because the majority of processing happens in the back end. The security issues that the researchers revealed are not business logic or application-specific issues. The issues are weaknesses in the application development, i.e., security tasks that developers should be doing, but they are not doing.

> **"Remediating the security vulnerabilities in the later stages of the software development life cycle is time consuming and very costly."**

Mobile applications that were tested during the research were leaking information through insecure coding. For example, these applications were vulnerable to:

- Man-in-the-middle (MitM) attacks
- Cross-site scripting (XSS) attacks
- Leakage of sensitive information through system logs
- Hard-coded credentials in the code
- Using non-Secure Socket Layer (SSL) (HTTP) protocol for transmitting sensitive information between the remote server and the user device

Recommendations from the researchers to improve the security of mobile applications include:[13]

- Ensure that all connections between the mobile applications and the back-end servers are performed using SSL. SSL certificate check is enforced by the client application to safeguard against interception and MitM.
- Protect the sensitive data that are stored in the device (client side) with encryption.
- Use code obfuscation and anti-debugging tricks to deter attackers from reverse engineering the application binary.
- Enable the protections that are provided by the mobile operating platform, such as:
  - Automatic reference counting (ARC)
  - Position-independent executable (PIE)
  - Stack protection in iOS platform
  - Using the latest software development kit (SDK)
  - Disabling the debugging feature in the compiled binary
  - Permission hardening for Android-based applications

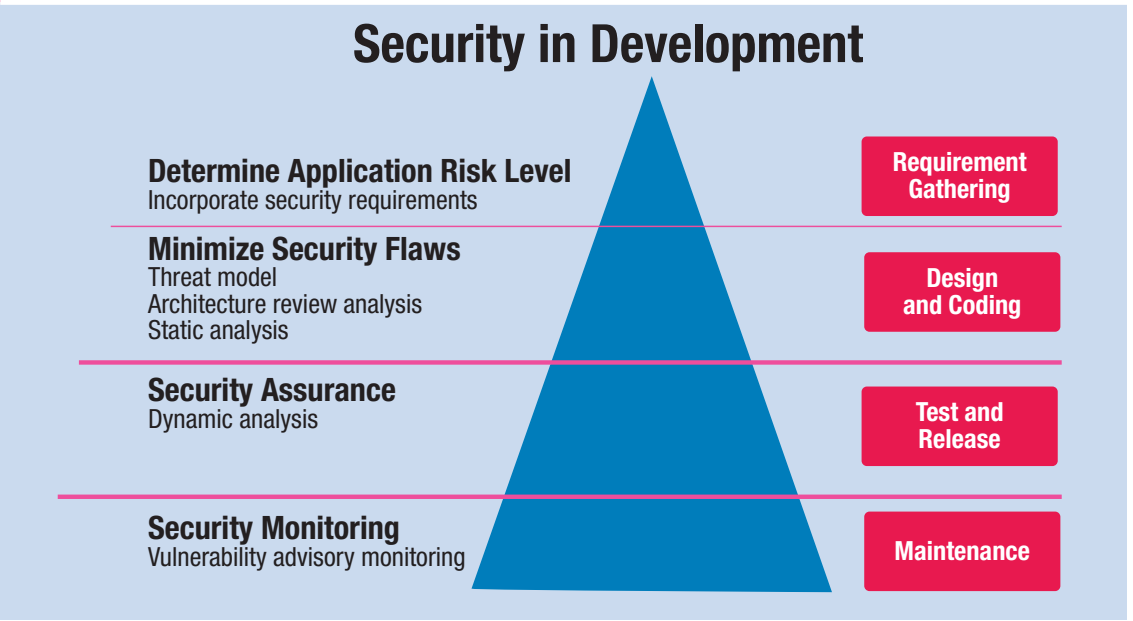
Building Security in Development

Performing application security assessments and incorporating security in the application just prior to the release of the software is not an ideal approach. Remediating the security vulnerabilities in the later stages of the software development life cycle (SDLC) is time consuming and very costly.

Secure development life cycle aims to incorporate security in all phases of software development, from requirement gathering to testing, release and maintenance (**figure 1**).[14, 15]

The following sections aim to provide guidance to application development and security personnel for embedding specific information security activities in each phase of the SDLC.

Figure 1—Embedding Security in Mobile Application Development



Source: S. Rajendran. Reprinted with permission.

Requirement Gathering

Embedding security in application development begins at the requirement gathering phase. Apart from business functionality requirements of the software, determine the:

1. User-specific security requirements expected in the application. This can include confidentiality, integrity, availability and authentication.
2. Importance of the data handled by the application and security requirements to protect the data
3. Compliance and regulatory mandates that are applicable for the users, the region where the application will be used and the information that is handled by the application
4. Use and misuse cases from a security perspective
5. Requirement traceability matrix to map requirements with security risk

Design

In the application design phase, the functional requirements are converted to architecture. It is important to embed security controls for application security in the design phase. Constructing a secure design minimizes the majority of security issues, because code-level issues can be identified with static analysis or manual code review. Furthermore, automated tools cannot identify design inconsistencies unless efforts are made to do a threat-model and architecture review.[16] Strict adherence to secure design principles greatly improves security.

Recognizing the importance of design in the security of applications, the Institute of Electrical and Electronics Engineers (IEEE) launched the Center for Secure Design (CSD) initiative. CSD identified the top 10 design flaws and ways to avoid them.[17] The CSD recommendation can provide valuable guidance to consider in the design of an application.

Performing threat modeling and architecture risk analysis of the design gives a measure of how likely it is that the software will be attacked and the extent of damage that an attack could cause. Start the analysis by building a high-level overview of the proposed system; then, analyze the design from an attacker's perspective, i.e., find ways to exploit the application.

Coding

During the coding phase, business/customer/product requirements are converted into an application. The input for this phase comes from the

Enjoying this article?

- **Learn more about, discuss and collaborate on mobile computing in the Knowledge Center.** *www.isaca.org/mobile-computing*

previous phases in SDLC (requirement gathering and design). Developers convert the design documents into functioning software. Incorrect writing of a code results in software errors. Coding errors can be reduced greatly when secure coding guidelines are applied in application development.

Coding guidelines can be either of the following:

- Generic, which are applied in all development environments irrespective of the platform chosen to construct an application. The Open Web Application Security Project (OWASP) Mobile Security Project[18] and the European Union Agency for Network and Information Security (ENISA) secure mobile application guidelines[19] are generic guidelines.
- Platform-specific coding guidelines related to a development platform, e.g., Android[20] or iOS[21]

> **"Constructing a secure design minimizes the majority of security issues, because code-level issues can be identified with static analysis or manual code review."**

Use of Third-party Code

Another important consideration during coding is the use of development frameworks and third-party libraries, including open-source components. Today, many applications are assembled out of multiple sets of libraries, most of which are open source, allowing the developer to focus on the core application functions while relying on third-party code to provide supporting capabilities. Although this is beneficial to develop the functionality quickly, some security breaches have happened due to vulnerabilities found in the libraries. Examples include the OpenSSL flaw that led to the Heartbleed vulnerability.[22]

It is recommended to create an inventory of open-source and third-party libraries that are used in the application that is being developed and retain the inventory as part of the development artifacts. Because open source comes from multiple parties and is introduced in the application code by developers from in-house and/or outsourced partners, it is essential that the inventory tracks the open-source component in the code and determines if these components are affected by known vulnerabilities.

A benefit of open-source inventory is that when any security incident takes place involving these libraries, remediation can be very quick, especially when the enterprise has several applications in its portfolio. A lack of information on open-source components that are used in applications can make it difficult to initiate remediation activities.[23]

Another advantage of open-source inventory is proactive monitoring of vulnerabilities in open-source components by referring to the inventory sheet (**figure 2**) and taking appropriate corrective action when something undesirable is forthcoming. Heartbleed led to a crisis situation for those who used OpenSSL cryptolibraries in their mobile applications, and updating to the current version was a challenge.[24, 25] In circumstances such as this, having an inventory sheet is helpful. The personnel who are responsible for support and maintenance know the details of applications and can use the inventory sheet to find where the vulnerable component is in use and then plan for remediation.

It is also worthwhile to vet the open-source and third-party libraries. The objective of vetting them is to minimize vulnerabilities, e.g., backdoors embedded in them or other security issues. Securing third-party open-source code can be approached in two ways: by embedding administrative controls and by embedding technical controls throughout the SDLC.

The first approach consists of administrative controls, such as policies and procedures. This approach can include:

- Developer awareness training to educate how developers inadvertently inherit security risk from

Figure 2—Sample Inventory Template for Third-party and Open-source Code

Functional module within the application where third-party library is used	Third-party library name	Third-party code version and patch level in use	Third-party code vendor/supplier	From where the third-party code component obtained	Most recent component version and release date (security updates)	Validation of third-party code for existence of vulnerabilities (indicate reference to verification information or defect tracking, if open items)

Source: S. Rajendran. Reprinted with permission.

open-source components to their application when the third-party code is not validated

- Audit of any open-source software in use, especially in high-priority applications
- Creation and maintenance of a list of approved/white-listed open-source code and restricted usage of unapproved software. However, white-listing may not be helpful when the volume of applications that an enterprise releases is high and when there is an increased need for the use of third-party code. In such situations, combining a white-listing approach with technical controls can help in achieving a fine balance.

The second approach consists of technical controls and performing source-code analysis and run-time analysis on the third-party code using automated tools. All third-party code that is used in the application should be subjected to these analyses to make sure that the potential security risk is identified and managed appropriately.

Freely available tools, such as Androwarn,[26] LinkedIn Quick Android Review Kit (QARK),[27] FindBugs[28] and Facebook Infer[29] can be used for analyzing the code.

Static Application Security Testing

Running static analysis on the source code early in the life cycle helps to fix code-level bugs before the application is released for general use. Static analysis finds incorrect coding that can potentially cause security risk. Analysis is performed without actually executing the program. The entire source code or binary is covered in this kind of analysis. It can be built in the development process and performed early in the software development life cycle.

Developers can be empowered to perform static analysis of their code and fix incorrect coding regularly. Integrating static analysis with continuous integration servers, e.g., Jenkins, minimizes the need for manual intervention, reduces dependency on the security team and fixes bugs that might turn into security vulnerabilities before they become unmanageable. Security tools, such as Androwarn, QARK, FindBugs and Infer, can be used for this analysis as well.

Developer Training

An IT organization striving to deliver secure applications (including mobile) must engage its developers and train them in secure coding practices. The focus must include delivering

a security risk–free application apart from the functionalities and features.

For example, Damn Vulnerable iOS App (DVIA),[30] as the name suggests, is a vulnerable mobile application. The main objective of the application is to teach developers and security enthusiasts about vulnerabilities in iOS mobile applications, based on the OWASP "Top 10 Mobile Risks."[31] Similarly, OWASP GoatDroid provides a training environment for Android developers and testers.

> **"In the testing phase, it is important to perform security testing along with quality assurance (QA) tests to continuously integrate security into development."**

Testing

In the testing phase, it is important to perform security testing along with quality assurance (QA) tests to continuously integrate security into development. QA assures the quality of the application to deliver the needed business functionality. Security tests give an assurance that the application is securely processing the business information.

Dynamic Application Security Testing (DAST) or run-time analysis is appropriate in this phase of the SDLC. Dynamic analysis is performed against a running instance of a program. This test most accurately mimics how a malicious user can attack the application.

Similar to traditional web application security testing, mobile application assessment requires a testing environment to effectively carry out the assessment. However, the security testing environment for mobile varies because the assessment involves reviewing multiple components, including how the application behaves when installed on the mobile device.

Setting Up a Mobile Testing Lab

A mobile testing lab requires the following essentials:

- A network connection. This environment must be isolated from the corporate or production network. Creating a Wi-Fi hotspot using a 3G/4G data card is an option. It is important to remember that both the analysis laptop and the device in which the mobile application is installed need to connect to the same network for some of the tests.
- A Mac or Windows laptop loaded with open-source security software
- A jailbroken[32] device for iOS application security testing (iPhone, iPod or iPad)
- For Android devices, an Android SDK and Eclipse integrated development environment (IDE) to set up an emulator[33]

Minimum Baseline Security Test Cases

Four major components of the mobile application environment need to be covered in the dynamic analysis:

- Device where the mobile application is installed
- Application
- Network communication between the application and enterprise server
- Data handled in the application

When establishing mobile application security testing capability, it may not be possible to focus on everything. The best approach is to start small and iterate continuously to mature the capability, incorporating lessons learned into the process along the way. The OWASP "Top 10 Mobile Risks" can be a good starting point when building the test cases for mobile security testing. Security professionals who are involved in application security voluntarily contribute to OWASP, which fairly represents the major security issues with mobile applications.

Alternatively, mobile application security test cases can be built based on the five security issues highlighted in the Hewlett-Packard "Mobile Application Security Study" report.[34] The security issues identified in the study are an abridged version of the OWASP Top 10, because study results are mapped with OWASP top issues.

Previous security assessment results of applications developed/used in the enterprise are other valuable resources to consult when building the test cases.

Attempts to establish a minimum baseline of security test cases can result in identifying high-level security objectives. These objectives are unique and relevant to mobile application security, as shown in **figure 3**.

The next step is to divide the security objectives into actionable security test cases. Mapping the security test cases with security assessment tools is another subactivity in this effort. Commercial security tools for mobile application assessment may not cover all of the test scenarios. Performing manual testing with some of the freely available open-source tools can give reasonable coverage to identify security risk.

Figures 4 and **5** break down security test objectives into test cases and map them to security assessment tools for iOS and Android mobile platforms.

Maintenance

Application security is an ongoing task; it continues to be important even when the application is released for public use. Proactively monitoring the security vulnerabilities in platform system software and embedded components and then initiating incident response and remediation, as appropriate, are crucial.

Identifying security vulnerabilities using reputable sources for obtaining security information is a continuous cycle. Sources such as software vendor websites, the US National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD), and the MITRE Corporation Common Vulnerabilities and Exposures (CVE) are reliable for vulnerability research.

Inventory of all third-party frameworks/APIs that are used in the mobile application is helpful to handle security patches. Whenever any vulnerability

Figure 3—Security Objectives Relevant to Mobile Application Security

Security Objective	Coverage	Purpose
Insecure data storage	Device	To find storage of credentials in property list files or SQLite database
Run-time manipulation	Application	To determine if the application is susceptible to modification of input to be interpreted as a code instruction
File system analysis	Application	To analyze if any sensitive application data are stored insecurely in the device
Analyzing network traffic	Network	To determine whether the application trusts any SSL certificate presented while connecting with enterprise IT infrastructure, resulting in MitM attacks
Insecure or broken cryptography	Application	To find out whether a weak or flawed encryption algorithm is used to secure the information
Information disclosure	Application	To find information leakage via logging, sending analytics data to external providers
Improper session handling	Application	To find out whether session timeout is set in the application
Binary protection	Data	To determine whether the mobile application binary is secure from reverse engineering risk
Privacy violations	Data	To find out whether the application is using more permission than necessary to collect and transmit user personal data elsewhere
Authentication	Network, application	To determine whether authentication is performed on the server side rather than on the client side (device)

Source: S. Rajendran. Reprinted with permission.

becomes public knowledge, a corresponding security update must be done for the mobile applications that are using these vulnerable third-party APIs/frameworks.

> **"Application security is an ongoing task; it continues to be important even when the application is released for public use."**

Conclusion

Mobile and web applications dealing with sensitive, private or other at-risk information require a secure development life cycle. Applications without security considerations may present an unexpected vulnerability to privacy. To address application security issues, developers are encouraged to understand the potential risk for each business function, code change and use of third-party frameworks and APIs, while security teams can help to improve application security through training, periodic scanning, ongoing vulnerability assessments and proactive engagement with developers. Incorporating security in all phases of SDLC instead of incorporating security just

Figure 4—iOS Security Test Cases

ID	Test Name	Security Tools to Use
IOS-01 IOS-02 IOS-03 IOS-04	Storage of credentials in property list file Storage of credentials in SQLite file Failure to use keychain to store credentials Storage of sensitive application data on file system	PuTTY, WinSCP, iExplorer, Plist Editor
IOS-05 IOS-06 IOS-07	Client trusting any SSL certificate presented—expired or invalid Application allows trivial MitM attack Connect to HTTPS once and fall back	Burp Proxy, Fiddler
IOS-08	Application logging sensitive application data	iPhone Configuration Utility
IOS-09	Application is storing its image in a public folder rather than application sandbox (application backgrounding)	PuTTY, WinSCP, iExplorer
IOS-10 IOS-11	Analytics data sent to third parties Authentication requests are performed on server side	Burp Proxy, Fiddler
IOS-12	Persistent authentication, if implemented, does not store user password on the device	WinSCP, Python, BinaryCookieReader.py
IOS-13	Hard code of cryptokeys in any construct (plain text, property files, compiled binaries)	IDA, Clutch, Class-dump-z
IOS-14	Use of insecure and/or deprecated algorithms	
IOS-15	Use of custom encryption protocols	
IOS-16	Invalidate sessions on the back end	Burp Proxy, Introspy
IOS-17	Reset cookies during authentication state changes	
IOS-18	Adequate time-out protection on the back-end components	
IOS-19	Code obfuscation	IDA, Clutch, Class-dump-z
IOS-20	Remove debugging statements and development information	iRET
IOS-21	Implementation of address space layout randomization (ASLR) PIE and automatic reference counting	iRET
IOS-22	Privacy violations—access to location, contacts, address book, photos	Snoop-it, iRET
IOS-23	Access to private data	Snoop-it, iRET
IOS-24	Run-time analysis	GDB, IDA, Hopper, ClutchMod

Source: S. Rajendran. Reprinted with permission.

Figure 5—Android Security Test Cases		
ID	**Test Name**	**Security Tools to Use**
AN-01 AN-02 AN-03 AN-04	Storage of credentials in device Storage of credentials in SQLite file Failure to use keystore to store credentials Storage of sensitive application data on file system	AXMLPRINTER, SQliteSpy, Cookies Manager+
AN-05 AN-06 AN-07	Client trusting any SSL certificate presented—expired or invalid App allows trivial MitM attack Connect to HTTPS once and fall back	Burp Proxy, Fiddler
AN-08	Application is logging sensitive application data	Burp Proxy
AN-09 AN-10	Analytics data sent to third parties Authentication requests are performed on the server side	Burp Proxy, Fiddler, secure code review
AN-11	Persistent authentication, if implemented, does not store user password on the device	Manually review
AN-12	Hardcode of cryptokeys in any construct (plain text, property files, compiled binaries)	Dex2Jar, JD-GUI, FindBugs, Androwarn
AN-13	Use of insecure and/or deprecated algorithms	
AN-14	Use of custom encryption protocols	
AN-15	Invalidate sessions on the back end	Burp Proxy, Introspy
AN-16	Reset cookies during authentication state changes	
AN-17	Adequate time-out protection on the back-end components	
AN-18	Code obfuscation	Dex2Jar, JD-GUI, Proguard
AN-19	Remove debugging statements and development information	Dex2Jar, JD-GUI,
AN-20	Privacy violations—access to location, contacts, address book, photos	DroidBox, Drozer
AN-21	Access to private data	
AN-22	Run-time analysis	Drozer

Source: S. Rajendran. Reprinted with permission.

prior to release of the software not only benefits the organization from an economic and efficiency perspective, it also ensures that the business services are enabled securely.

Author's Note

The views expressed in this article are the author's and in no way represent the stance of his employer.

Endnotes

1 2014 Research Into Internet Systems LLC, "Top 10 Mobile Security Risks," Decompiling Android, 2014, *www.decompilingandroid.com/mobile-app-security/top-10-mobile-security-risks/?_sm_au_=iHVjTnqfJSv0F6Nj*

2 Tung, L.; "Hackers Access 800,000 Orange Customers' Data," *ZDNet*, 3 February 2014, *www.zdnet.com/article/hackers-access-800000-orange-customers-data/*

3 TrustedSec, "CHS Hacked via Heartbleed Vulnerability," TrustedSec Update, 19 August 2014, *www.trustedsec.com/august-2014/chs-hacked-heartbleed-exclusive-trustedsec/*

4 Mumsnet Limited, "The Heartbleed Security Breed—And What To Do," mumsnet, *www.mumsnet.com/info/the-heartbleed-security-breach-to-do*

5 Paganini, P.; "Vulnerabilities in Alibaba Threatens Security of Million Users," *Security Affairs*, 11 December 2014, *http://securityaffairs.co/wordpress/31028/hacking/vulnerabilities-in-alibaba.html*

6 Mai-Duc, C.; "Alibaba Security Flaws Exposed Data on Millions of Users, Analysts Say," *The Los Angeles Times,* 10 December 2014, *www.latimes.com/business/technology/la-fi-tn-alibaba-security-breach-20141210 story.html*

7 Whittaker, Z.; “Kindle Security Vulnerability Can ‘Compromise’ Amazon Accounts,” *ZDNet,* 16 September 2014, *www.zdnet.com/article/kindle-security-vulnerability-can-compromise-amazon-accounts/*

8 Wallop, H.; “eBay Hacking: Online Gangs Are After You,” *The Telegraph*, 23 May 2014, *www.telegraph.co.uk/technology/internet-security/10849689/eBay-hacking-online-gangs-are-after-you.html*

9 Android Studio, “Shrink Your Code and Resources,” *http://developer.android.com/tools/help/proguard.html*

10 Hewlett-Packard Development Company L.P., *Mobile Application Security Study,* February 2014, *www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-1057ENW.pdf*

11 Sanchez, A.; “Personal Banking Apps Leak Info Through Phone,” *IOActive*, 8 January 2014, *http://blog.ioactive.com/2014/01/personal-banking-apps-leak-info-through.html*

12 Higgins, K. J.; ”Weak Security in Most Mobile Banking Apps,” *InformationWeek DarkReading,* 12 December 2013, *www.darkreading.com/vulnerabilities---threats/weak-security-in-most-mobile-banking-apps/d/d-id/1141054*

13 *Op cit,* Sanchez

14 Microsoft, “What Is the Security Development Lifecycle?,” Security Development Lifecycle, *www.microsoft.com/en-us/sdl/*

15 BSIMM, “What We Do,” *www.bsimm.com/*

16 Sareen, P.; “Updated: After Ola & ZopNow Tech Screw Up, This Time Foodpanda Becomes the Target of a New Hack for Getting Free Food!!,” Inc42, 10 April 2015, *https://inc42.com/buzz/after-ola-zopnow-this-time-foodpanda-becomes-target-of-a-new-hack-for-getting-free-food/*

17 IEEE Cybersecurity, “Avoiding the Top 10 Software Security Design Flaws,” 13 November 2015, *http://cybersecurity.ieee.org/center-for-secure-design/avoiding-the-top-10-security-flaws.html*

18 The Open Web Application Security Project, “OWASP Mobile Security Project,” 18 July 2016, *www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Mobile_Security_Testing*

19 Euopean Union Agency for Network and Information Security, “Smartphone Secure Development Guidelines,” 25 November 2011, *www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/smartphone-security-1/smartphone-secure-development-guidelines*

20 Android, “Security Tips,” *http://developer.android.com/training/articles/security-tips.html*

21 Apple Inc., “Introduction to Secure Coding Guide,” Mac Developer Library, *https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html*

22 For more information, see *http://heartbleed.com/*

23 BlackDuck, “Future of Open Source Survey,” 2016, *https://info.blackducksoftware.com/rs/872-OLS-526/images/FOOS_Infographic_Security.pdf*

24 Helppi, V.; “What Heartbleed Bug Means to App Developers? Testdroid Has You Covered,” bitbar.com, 10 April 2014, *http://bitbar.com/what-heartbleed-bug-means-to-app-developers-testdroid-has-you-covered/*

25 Acharya, S.; “Heartbleed Bug: How to Protect Android Devices,” *International Business Times*, 12 April 2014, *www.ibtimes.co.uk/heartbleed-bug-how-protect-android-devices-1444508*

26 GItHub, “Androwarn,” *https://github.com/maaaaz/androwarn*

27 GItHub, “Qark,” *https://github.com/linkedin/qark*

28 GItHub, “findbugs,” *https://github.com/findbugs/findbugs*

29 GItHub, “Infer,” *https://github.com/facebook/infer*

30 Damn Vulnerable iOS Application (DVIA), *http://damnvulnerableiosapp.com/*

31 *Op cit,* OWASP

32 Gianchandani, P.; “iOS Application Security Part 1—Setting Up a Mobile Pentesting Platform,” 16 June 2013, *http://highaltitudehacks.com/2013/06/16/ios-application-security-part-1-setting-up-a-mobile-pentesting-platform/*

33 The Open Web Application Security Project, “SettingupMobileTestingLab,” 7 June 2013, *www.owasp.org/index.php/SettingupMobileTestingLab*

34 *Op cit,* Hewlett-Packard Development Company L.P.