Computer Networks and Communications
https://ojs.wiserpub.com/index.php/CNC

Article

A Framework for Security Assessment of Android Mobile Banking Applications

Loïc D. Tsobdjou*, **Samuel Pierre**, **Alejandro Quintero**

Department of Computer and Software Engineering, Polytechnique Montreal, Montreal, QC, Canada
E-mail: loic.tsobdjou-dongmo@polymtl.ca

Received: 16 November 2023; **Revised:** 1 February 2024; **Accepted:** 20 February 2024

Abstract: Mobile banking applications make users' daily lives easier by allowing them to access banking services, such as balance inquiries and bill payments, anytime and anywhere. Since these applications manage very sensitive financial data, special attention must be paid to data security. Several works in the literature assess the security of mobile banking applications. However, we observe the lack of a widely adopted framework among researchers for assessing the security of mobile banking applications. In this paper, we propose a framework consisting of twenty-six criteria for assessing the security of Android mobile banking applications. These criteria are divided into five categories: mobile device security, data in transit, data storage, cryptographic misuse, and others. Subsequently, we evaluate the proposed framework based on predefined requirements. These requirements are no redundancy, no ambiguity, and comprehensiveness. As a case study, we assess the security of the Android mobile banking applications of seven major Canadian banks. The results show that data in transit is adequately protected by these applications.

Keywords: Android, mobile banking, mobile code security, security assessment

1. Introduction

Mobile banking is the act of making financial transactions, such as balance inquiries, bill payments, and money transfers on mobile devices (e.g., smartphones, tablets) [1]. One of the benefits of mobile banking is access to banking services anytime, anywhere. However, its main disadvantage is the data security issue, as financial data is very sensitive. According to the Canadian Bankers Association, 86% of Canadians trust their bank to offer secure digital banking services [2]. The main factors that influence the use of mobile banking are the services offered and the security related to it [3]. Arisya et al. [4] revealed that mobile banking application users were aware of information security. Boutin and Chouinard [5] pointed out that cyberattacks in the Canadian financial sector have tripled between the years 2014 to 2019. For example, in May 2018, the leak of confidential personal data affected 40,000 and 50,000 customers of Canadian banks CIBC and BMO, respectively [5].

Service providers must then ensure data confidentiality, integrity, availability, and privacy. Data confidentiality refers to the fact that only authorized persons can access the data. Integrity ensures that the data is not altered and is in a usable format. Availability ensures that legitimate users can always access data. Privacy refers to protecting the users' identities.

Mobile devices face several security threats, including theft or loss, denial of service, malware, application vulnerabilities, and operating system vulnerabilities. A malicious user in possession of a stolen or lost mobile device, for example, can access the data stored on that device using forensic analysis techniques. This jeopardizes data confidentiality if the data is not well protected. Denial of service compromises data availability by preventing users from accessing their data and services. Malware acts without the user's knowledge to steal or alter data, thus affecting data confidentiality, integrity, and privacy. Application and operating system vulnerabilities are security holes in software. Moreover, cybercriminals can exploit these vulnerabilities to compromise data security.

In this paper, we focus on vulnerabilities in Android mobile banking applications. We chose the Android operating system for two reasons. The first reason is that Android is the leader in mobile operating systems worldwide, with about 70% of the market share in January 2022 [6]. The second reason is that the open nature of Android exposes it to more security threats.

Our main objective is to propose a framework for assessing the security of mobile banking applications in order to enable banks to integrate security considerations into applications' design and development.

The contributions of this paper are summarized as follows:

- We propose a set of concrete and comprehensive criteria for assessing the security of Android mobile banking applications.
- We define the requirements that the proposed set of criteria must satisfy, and we evaluate the proposed set of criteria based on these requirements.
- We assess the security of the Android mobile banking applications of seven major Canadian banks.

The rest of this paper is organized as follows. In Section 2, we present the background. In Section 3, we review the related works. We introduce the proposed framework and the requirements that this latter must satisfy in Section 4. In Section 5, we conduct a case study to assess the security of seven Android mobile banking applications from major banks in the Canadian market. Finally, we conclude our work in Section 6.

2. Background

This section consists of two sub-sections. The first sub-section presents the Open Web Application Security Project (OWASP) Mobile Top 10 project. The second one reviews the Android application security best practices defined by Google.

2.1 *OWASP Mobile Top 10*

The Open Web Application Security Project (OWASP) is a foundation launched in 2001 whose objective is to improve software security [7]. OWASP is a non-profit organization concerned with software security. Their projects and documents are free and open to anyone interested in improving application security. In addition, they are supported by renowned cybersecurity companies such as Tenable and NowSecure. In 2016, OWASP published a list of the ten most common security risks related to mobile applications under the project *OWASP Mobile Top 10* as follows [8]:

- M1: Improper platform usage;
- M2: Insecure Data Storage;
- M3: Insecure Communication;
- M4: Insecure Authentication;
- M5: Insufficient Cryptography;
- M6: Insecure Authorization;

- M7: Client Code Quality;
- M8: Code Tampering;
- M9: Reverse Engineering;
- M10: Extraneous Functionality.

The M1 risk refers to the misuse of mobile device operating system features. For example, using unnecessary Android permissions can lead to the leakage of sensitive user data. In addition, the use of application local storage instead of a keychain to store sensitive data in iOS applications.

The M2 risk occurs when sensitive data is stored without proper protection. As an illustration, the user password should not be stored in plaintext. We should protect the user password using a password-based key derivation function.

The M3 risk reflects the insecure transfer of sensitive data via a communication channel. It is important to mention that the communication channel between the mobile device and the server is not secure. A proper implementation of standard security protocols like Transport Layer Security (TLS) can be a solution to this risk.

In the M4 risk, malicious users can bypass the authentication process. As an example, a short password is prone to brute-force attacks.

The M5 risk refers to the misuse of cryptographic techniques. Vulnerabilities in this category include the use of encryption algorithms known to be insecure, the hardcoding of encryption keys in the source code of the mobile application, etc.

The M6 risk occurs when a user accesses features reserved for a higher-privilege user. Unauthorized access to a database can lead to data leakage and identity theft.

The M7 risk "captures bad code that executes on the mobile device itself" [8]. Examples include buffer overflows and untrusted inputs.

In the M8 risk, the source code of the mobile application is modified and redistributed to users. The repackaging attack is an example of this category. To prevent this risk, the mobile application can check its integrity at runtime.

The M9 risk refers to the generation and analysis of the source code of an application. It can be prevented by using obfuscation.

Finally, the M10 risk consists in "leaving functionality enabled in the application that was not intended to be released" [8].

2.2 *Google application security best practices*

In this section, we review some Android application security best practices. Google specifies these best practices [9] as the developer of the Android operating system. We distinguish four categories:

- enforce secure communication;
- provide the right permissions;
- store data safely;
- keep services and dependencies up-to-date.

The first category *enforce secure communication* concerns the security of communication between applications on the same device or between the application and a remote device like a server. The best practices in this category include:

- use implicit intents and non-exported content providers;
- ask for credentials before showing sensitive information;
- apply network security measures;

- use WebView objects carefully.

The second category *provide the right permissions* refers to the proper use of permissions. On Android, permission is a concept used to restrict the application's access to some data and resources on the mobile device. Google recommends using only those permissions that are necessary for the application to function properly. Best practices in this category are: (1) use intents to defer permissions; and (2) share data securely across applications.

The third category *store data safely* is to protect sensitive data at rest. Google recommends storing confidential data within internal storage, as other applications cannot access it, and the device deletes this data when the application is uninstalled. Moreover, sensitive data can be encrypted before being saved. Other best practices in this category include: (1) store data in external storage based on use case; (2) store only non-sensitive data in cache files; and (3) use *SharedPreferences* in private mode.

Finally, the fourth category *keep services and dependencies up-to-date* consists in updating all dependencies, such as libraries, before the application is released.

3. Related work

In this section, we review recent works assessing the security of mobile banking applications.

Chen et al. [10, 11] assessed the data-related weaknesses in Android banking applications. The study was based on 693 banking applications across eighty-three countries. The authors proposed an automated security risk assessment system (AUSERA) that identifies security weaknesses in banking applications. They claimed that their system outperforms four state-of-the-art industrial and open-source tools for data-related weakness detection, namely AndroBugs, Mobile Security Framework (MobSF), Quick Android Review Kit (QARK) and Qihoo 360. Security weaknesses were classified into four categories: input harvest, data storage, data transmission, and communication infrastructure. The authors found out that the most popular weaknesses of banking applications were screenshots, insecure hash functions, and inter-component communication (ICC) Leakage.

Bojjagani and Sastry [12] proposed a threat model that aims to detect and mitigate vulnerabilities in Android and iOS mobile applications. Moreover, they built a security testing framework to identify man-in-the-middle attacks in mobile applications. Finally, the compliance of mobile banking applications with the OWASP listed mobile security risks was analyzed. Mobile security threats were divided into three categories: application-level threats, communication-level threats, and device-level threats. Application-level threats are insecure data storage, lack of binary protection, unintended data leakage, malware in applications, weak cryptography, and privilege escalation. The communication level threat is insufficient transport layer protection. Device-level threats include mobile forensics and malware. For the security testing framework, the authors adopted both static and dynamic analysis. Static analysis examines the program structure, while dynamic analysis examines the program in operation. As case study, the authors analyzed five Android and three iOS mobile banking applications in India.

Zheng et al. [13] analyzed security vulnerabilities in Android mobile applications. They focused on repackaging attacks. In these attacks, an attacker disassembles a legitimate application, injects malicious code, and rebuilds it into a new application. The authors used three mobile banking applications in Australia as case study. They implemented repackaging attacks to steal sensitive information, such as Short Message Service (SMS) and contact lists. Some techniques to mitigate repackaging attacks are obfuscation, runtime detection, and improvement of user awareness.

Chanajitt et al. [14] analyzed seven Android mobile banking applications in Thailand. Their aim was to assess the security of these applications. More specifically, they analyzed the security of sensitive data after user registration, banking transactions, and the transition of data over the network. Moreover, they evaluated the resistance of the application against repackaging attacks. For the test, the authors used two Android devices: the Samsung Galaxy S4 GT-I9500 (rooted) and the Galaxy S4 mini-GT-I9190 (unrooted). Firstly, some activities were performed on the mobile banking applications. Then, the authors acquired an image of the flash memory for further forensic analysis. In addition, the application code was statistically analyzed to evaluate the security features.

Kaur et al. [15] assessed the security of the Android Pay application, along with the Android e-wallet applications of three Canadian banks. They focused on Host Card Emulation Near-Field communication (HCE-NFC) enabled applications. The authors proposed a set of security rules obtained from the Canadian Bankers Association and the OWASP Top 10 Mobile Risks. The set of rules consisted of device security (e.g., no rooted device and no emulator), application security (e.g., self-verification integrity, protected application, etc.), communication security and dynamic data (e.g., https enforced, proper certificate pinned, etc.), device storage (e.g., no plain text logs and no sensitive information in backups), and memory (e.g., no sensitive information in memory). The authors conducted both static and dynamic analysis. The results showed that e-wallet banking Android applications in the Canadian market were not well secured against some attack vectors, including rooted devices, emulators, and self-verification integrity.

Bojjagani and Sastry [16] analyzed Android mobile banking applications at the application level, network level, and device level. They identified some threat scenarios, namely untrusted party learning of bank data, tampering with bank data, and a customer choosing the wrong bank application. The mobile vulnerabilities were masquerade, man-in-the-middle, replay, traffic analysis/Wi-Fi sniffing, browser exploits, SQL injection, lack of binary protection, broken cryptography, data leakage, and insecure data storage. The testing strategy consisted of four parts: static analysis, dynamic analysis, Web application server security, and device forensic.

Jung et al. [17] studied repackaging attacks against Android banking applications in the South Korean market. Seven Android banking applications were analyzed. They implemented a repackaging attack in which money transfers were sent not to the intended recipient but to the attacker. The attack succeeded for all seven banking applications. As countermeasures to repackaging attacks, the authors suggested self-signing restrictions, code obfuscation, and code attestation.

Reaves et al. [18] investigated the security of branchless banking applications (also known as mobile money applications) in developing countries. Firstly, the authors analyzed forty-eight Android mobile money applications from twenty-eight countries using the automated tool Mallodroid. This is a static analysis tool for detecting TLS vulnerabilities. Then, the entire application communication flow analysis was performed on seven applications from Brazil, India, Indonesia, Thailand, and the Philippines. This analysis consisted of two phases. Phase 1 was the inspection to identify the basic functionality of the application, whereas phase 2 was the reverse engineering method. The authors were interested in security errors in the following procedures: registration and login, user authentication after login, and money transfers.

Bassolé et al. [19] discussed the security of mobile banking and mobile payment applications. Specifically, they analyzed vulnerabilities in Android mobile banking and mobile payment applications in African countries. The case study was based on fifty-three mobile applications from African banks or banks with subsidiaries in Africa. The analysis procedure consisted of two steps, namely the static analysis and the dynamic analysis. For static analysis, the authors used Apktool for reverse engineering, whereas for dynamic analysis, they used the virus-total platform for malware detection.

Castle et al. [20] addressed the question of security vulnerabilities in digital financial services (DFS) in developing countries. They first defined a threat model. Then, they analyzed the security of 397 DFS deployments, including 197 Android applications. Finally, they interviewed seven developers and product managers. In the threat model, the security goals were confidentiality, integrity, and availability. Potential adversaries were customers, agents (i.e., intermediaries between users and their accounts), and organization employees. Potential attacks included external applications, man-in-the-middle, authentication attacks, and denial of service. For security analysis, the authors considered information leakage, Android permissions, the Uniform Resource Locator (URL) and third-party libraries.

Chothia et al. [21] analyzed TLS related vulnerabilities in mobile banking applications in the United Kingdom (UK). As case study, they examined Android and iOS applications from fifteen banks in the UK. The most common TLS flaws were self-signed certificates, no hostname verification, protocol downgrades or weak cipher suites, and Secure Sockets Layer (SSL) stripping. The first stage of the investigation was to look at well-known TLS vulnerabilities using BurpSuite, Mallodroid and SSLStrip tools. Furthermore, the authors analyzed the vulnerability of using certificate pinning without hostname verification.

Darvish and Husain [22] discussed the security of Android mobile banking and mobile payment applications. They performed both static analysis and dynamic analysis on twenty-six applications. The authors used AndroBugs, APKtool, and Moblizer tools for static analysis and Charles Proxy and BurpSuite tools for dynamic analysis. The security analysis highlighted some vulnerabilities, such as insecure data storage and SSL certificate verification.

Osho et al. [23] conducted a forensic analysis of twelve Android mobile banking applications in Nigeria. This analysis was done based on the five requirements defined by OWASP's Mobile AppSec Verification Standard-Level 2 (MASVS-L2). As a result, none of the analyzed applications saved sensitive data in the generated backup, and none of them removed sensitive data when the application switched to the background.

Uduimoh et al. [24] conducted a forensic analysis of five Android mobile banking applications in Nigeria. The results revealed that all analyzed mobile banking applications saved sensitive data in plaintext.

Table 1 classifies related work according to the type of analysis and operating system.

Table 1. Classification of related work

Related Work	Static Analysis	Dynamic Analysis	Automatic Analysis	Manual Analysis	Operating System
[10, 11]	Yes	No	Yes	Yes	Android
[12]	Yes	Yes	Yes	Yes	Android iOS
[13]	No	Yes	No	Yes	Android
[14]	Yes	Yes	No	Yes	Android
[15]	Yes	Yes	Yes	Yes	Android
[16]	Yes	Yes	Yes	Yes	Android
[17]	No	Yes	No	Yes	Android
[18]	Yes	Yes	Yes	Yes	Android
[19]	Yes	Yes	Yes	Yes	Android
[20]	Yes	Yes	No	Yes	Android
[21]	Yes	Yes	Yes	Yes	Android iOS
[22]	Yes	Yes	Yes	No	Android
[23]	No	Yes	No	Yes	Android
[24]	No	Yes	No	Yes	Android

The observation made at the end of this literature review is the lack of a widely adopted framework among researchers for assessing the security of mobile banking applications. We propose a solution to this problem by defining a concrete and comprehensive set of criteria for the security assessment of mobile banking applications.

The similarity between related work and ours is the security assessment of mobile banking applications. Through dynamic analysis, we assess the security of data in transit. However, unlike existing work, we focus on Android mobile banking applications in Canada.

Furthermore, the main objective of this paper is to propose a framework for assessing the security of Android mobile banking applications. This is the main difference with related works that aimed to design automatic tools for the analysis of mobile banking applications, or to evaluate the security of mobile banking applications based on criteria defined by each author. To the best of our knowledge, this is one of the first works to focus on defining a framework that can serve as a common basis for assessing the security of Android mobile banking applications.

4. Proposed framework

4.1 *Requirements*

In this section, we define the requirements of the proposed framework. These requirements are inspired by the work presented in [25]. Wang and Wang [25] proposed a framework for evaluating two-factor authentication schemes that includes a set of twelve criteria. In our framework, we suggest a set of twenty-six criteria for the security assessment of mobile banking applications. We used the same requirements as those presented in [25] since, to the best of our knowledge, there is no previous work that defines the requirements of a framework for security assessment of mobile banking applications. The requirements of our framework are as follows:

- No redundancy. We should have a unique meaning for each criterion within a proposed set of criteria. In addition, a given criterion should not be included in another. The requirement of non-redundancy avoids having an unnecessarily long list of criteria.
- No ambiguity. This requirement refers to the fact that each criterion must have a unique interpretation in the context of assessing the security of Android mobile banking applications. Moreover, each criterion must allow for the assessment of a mobile banking application individually without referring to another mobile banking application.
- Comprehensiveness. The proposed set of criteria must capture a wide range of security vulnerabilities faced by Android mobile banking applications. Vulnerabilities can arise from insecure storage of sensitive data, insecure exchange of sensitive data with a remote server, misuse of cryptography, etc.

4.2 *Criteria set*

In this section, we define a set of criteria to help evaluate the security of Android mobile banking applications. For this purpose, we integrate security vulnerabilities from previous research [10, 12, 14, 15, 16, 18, 20, 21, 26, 27] and industrial reports [8, 9, 28, 29]. First, we identified the security vulnerabilities cited in previous work. Next, we eliminated redundant vulnerabilities. Finally, we ensured that the selected vulnerabilities covered all the security risks listed in the OWASP Mobile Top 10. The result is a set of twenty-six criteria that mobile banking applications should satisfy to ensure good security. These criteria are summarized in Table 2.

Table 2. Criteria Set

Category	Criterion	Reference
Mobile device security	No rooted device	[15]
	No emulator	[15]
Data in transit	Reject self-signed certificate	[21]
	Verify certificate hostname	[21]
	Reject expired or revoked certificate	[10]
	Transmit sensitive data in encrypted form	[18]
Data storage	No sensitive data in plaintext in internal storage	[9]
	No sensitive data in plaintext in external storage	[10]
	No sensitive data in plaintext in log files	[10]
	No sensitive data in plaintext in cache files	[9]
	Use *SharedPreferences* in private mode	[9]
Cryptographic misuse	No hard-coded decryption key or other credentials	[18]
	No ECB mode for encryption	[26]
	Use a random IV for CBC encryption	[26]
	Use a random salt for PBE	[26]
	Use more than one thousand iterations for PBE	[26]
	Use a secure random value	[10]
	Use a secure encryption algorithm or hash function	[10]
Others	Disable UI screenshots	[10]
	Obfuscation	[29]
	No debuggable	[15]
	No third-party ads libraries	[15]
	Limit login attempts	[8]
	Integrity check at runtime	[13]
	Disable access to application's content providers	[12]
	Control of method inputs	[8]

We distinguish five categories: (1) mobile device security; (2) data in transit; (3) data storage; (4) cryptographic misuse; and (5) others.

The first category, *mobile device security,* refers to the detection of insecure devices. The two criteria in this category are: (1) no rooted device; and (2) no emulator. It is necessary for sensitive applications like mobile banking applications to verify the integrity of the mobile device at runtime. Indeed, rooted devices and emulators are sources of security vulnerabilities. A rooted device is an Android device whose user has superuser rights. Thus, any application on the device can perform dangerous actions like deleting system files. In addition, access to data and resources on the device is no longer restricted. An emulator is a virtual machine that simulates a real Android device. The behavior of an application running on an emulator can be analyzed by a hacker [15].

The second category, *data in transit,* contains criteria related to the protection of sensitive data when it is transmitted over the communication channel. The objective is to check the vulnerability of the applications to network attacks, such as man-in-the-middle attacks and server spoofing. Data in transit faces security risks associated with the communication network between the application and the server. We focus on the TLS protocol since it is the standard used to secure communications on the Internet. The criteria in this category assess the proper implementation of the TLS protocol within the application as follows:

- Reject self-signed certificate. An application should not accept communication with a server that presents a self-signed certificate. By doing so, it accepts communication with any server, including a server controlled by a hacker. The application can then transmit sensitive data to this malicious server. The application should only communicate with a server that has a certificate signed by a trusted authority.
- Verify certificate hostname. The application must verify that the requested hostname matches the one of the received certificates. Without hostname verification, the application may accept communication with a server that has a valid certificate signed by a trusted authority. However, it might not be the server the application is supposed to communicate with. In this case, the application may transmit sensitive data to an unknown entity.
- Reject expired or revoked certificates. This criterion verifies that the application accepts only certificates that have not passed their expiration date and have not been revoked. The failure to do this check may result in communication with a server that has been revoked and is now under the control of a hacker.
- Transmit sensitive data in encrypted form. Sensitive data must be encrypted before being transmitted over the communication channel. This can be done using the TLS protocol. Moreover, data can be encrypted before being encapsulated by the TLS protocol to provide an additional layer of security. Furthermore, we use the protocol HyperText Transfer Protocol Secure (HTTPS) instead of HyperText Transfer Protocol (HTTP) to communicate securely.

The third category of criteria is data storage. The aim is to verify that the sensitive data on the mobile device is properly protected. The sources of attacks can be malware or a hacker in possession of a stolen or lost mobile device. The criteria in this category are listed below:

- No sensitive data in plaintext in internal storage. In the Android operating system, the data stored in the internal memory is sandboxed by the application [9]. Thus, a given application cannot access the data of another application. However, this data can be recovered by a hacker in possession of the device using forensic tools. Therefore, it is recommended to always encrypt sensitive data before storing it in internal storage. Hence, this data is protected from any entity that does not have the decryption key.
- No sensitive data in plaintext in external storage. As much as possible, a mobile banking application should avoid saving sensitive data in external storage since it is accessible by other applications, including malware. When external storage is used, the data must be encrypted beforehand. Moreover, the availability of external storage should be checked since it is removable. In addition, the validity of the data must be checked, as the data can be corrupted or modified by other applications [9].
- No sensitive data in plaintext in log files. During application development, developers usually use logs for testing purposes. Sometimes, these logs remain when the application is deployed. The logs generated by an application can constitute a security vulnerability if they contain sensitive data in plaintext.
- No sensitive data in plaintext in cache files. The cache memory is often used to allow the application to access data quickly. This memory should only be used for non-sensitive data [9].

- Use SharedPreferences in private mode. According to Google's recommendation [9], SharedPreferences objects should be used in private mode. Therefore, other applications will not have access to the data stored in these objects.

The fourth category, cryptographic misuse, evaluates the implementation of cryptographic techniques within the mobile application. Cryptography can be used to protect sensitive data. Moreover, it can be used as an additional security layer for the transmitted data over a communication channel. In this case, the data is encrypted before being encapsulated by the TLS protocol. However, some rules must be respected so that a hacker in possession of the encrypted data cannot retrieve the plaintext data. These rules are presented as follows:

- No hard-coded decryption key or other credentials. If the decryption key and other credentials, such as the credentials to access a database, are hard-coded in the application, they can be recovered with reverse engineering tools. The hacker can then decrypt the encrypted data or access the database. It is important not to hardcode the decryption keys and other credentials.
- No ECB mode for encryption [26]. Electronic Code Book (ECB) is a mode of operation used by block symmetric encryption. In this mode, given an encryption key, a block of plaintext will always produce the same block of ciphertext [30]. This mode of operation is not recommended for sensitive applications, such as mobile banking applications.
- Use a random IV for CBC encryption [26]. Cipher Block Chaining (CBC) is another mode of operation used by block-symmetric encryption. This mode of operation requires an initialization vector (IV) to generate the ciphertext from the plaintext. This initialization vector must be unpredictable to avoid attacks on the encryption algorithm [30].
- Use a random salt for PBE [26]. In Password-Based Encryption (PBE), we use the notion of salt to counter dictionary and rainbow table attacks. The salt must be random and different for each password to avoid two identical passwords having the same hash. Having a constant salt is the same as having no salt, because in both cases, two identical passwords will always have the same hash.
- Use more than one thousand iterations for PBE [26]. The notion of iteration is used in the PBE to counter brute-force attacks. As per the National Institute of Standards and Technology (NIST) recommendation, "the iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users" [31]. The minimum recommended value is one thousand.
- Use a secure random value. Random values used for cryptographic purposes must have high entropy. Android provides the SecrureRandom class to generate secure random numbers.
- Use a secure encryption algorithm or hash function. Some encryption algorithms and hash functions have known security flaws. They should no longer be used to protect sensitive data. Examples of insecure encryption algorithms and hash functions are RC2, MD4, MD5, SHA1 [32].

The last category, others, contains various criteria that do not fit into any of the above presented categories. These criteria are presented as follows:

- Disable UI screenshots. Malware can take screenshots to collect sensitive data like passwords when entered by the user. Mobile banking applications must prevent user interface (UI) screenshots.
- Obfuscation. Obfuscation is a technique that makes it difficult to reverse engineer the application. Obfuscation is, therefore, useful to protect the source code of the application.
- No debuggable. If the application is debuggable, then it can be connected to a debugger to analyze the application's behavior step by step [15]. The android:debuggable attribute present in the Android manifest allows to determine if an application is debuggable (true value) or not (false value).

- No third-party ads libraries. Since a mobile banking application is a sensitive application, it should not use third-party ads libraries. This is because third-party ads libraries can be sources of security vulnerabilities [15].
- Limit login attempts. To prevent brute force attacks, the application must limit the number of incorrect login attempts for a single user.
- Integrity check at runtime. At runtime, the application must check that it has not been modified. This prevents the repackaging attack, where a hacker injects malicious code into the original application.
- Disable access to the application's content providers. According to Google's recommendation [9], access to the ContentProvider object must be disabled so that other applications cannot access it.
- Control of method inputs. All method inputs should be considered untrusted. Therefore, it is important to check the inputs to prevent attacks like code injection.

In this set of criteria, we did not consider the use of unnecessary permissions because this criterion depends on the services offered by each application. Indeed, applications that have the remote cheque deposit feature require access to the smartphone camera to scan a cheque, while other applications would not. Furthermore, we do not provide weights to the different criteria since a bank may decide to give preference to certain criteria over others, depending on its specific goals.

4.3 *Evaluation*

In this section, we show that the proposed framework satisfies the defined requirements: no redundancy, no ambiguity, and comprehensiveness.

- No redundancy. Chen et al. [10] proposed the most recent set of security weaknesses of the mobile banking applications. The weakness stored on SD card can be confused with the weakness written in text files. Indeed, when sensitive data are written in text files, these text files are stored either in the internal memory in mobile device or in the external memory (e.g., an SD card). If the file is stored on an SD card, then the weaknesses stored on the SD card and written in text files are identical. Furthermore, the weaknesses of using invalid certificates and invalid certificate authentication have the same meaning. Hence, the redundancy is in this set of security weaknesses. To avoid redundancy in our set of criteria, we have divided the criteria into five categories. Anyone can check that no one criterion has the same meaning as another or is included in another criterion.
- No ambiguity. To eliminate ambiguity in the proposed framework, we have clearly described each criterion in Section IV-B. As a result, each criterion has a unique interpretation. Furthermore, each criterion allows for the assessment of a mobile banking application individually without referring to another mobile banking application.
- Comprehensiveness. The proposed set of criteria covers all the risks listed in the *OWASP Mobile Top 10* project. The criterion of *no hard-coded decryption key or other credentials* refers to the M1 risk. In this case, the application developer hardcodes the decryption keys instead of using the Android Keystore System, which offers better protection of cryptographic keys. The criteria of the category *data storage* cover the M2 risk, while the M3 risk is covered by the criteria of the category *data in transit*. The criteria corresponding to the M4 and M6 risks are *reject self-signed certificate*, *verify certificate hostname*, *reject expired or revoked certificate*, *no hard-coded decryption key or other credentials,* and *limit login attempts*. The criteria in the category of *cryptographic misuse* cover the M5 risk. The criterion *control of method inputs* refers to the M7 risk. The criterion *integrity check at runtime* deals with the M8 risk, while the criterion *obfuscation* deals with the M9 risk. Finally, the M10 risk is addressed by the criteria of *no sensitive data in plaintext in log files* and *no debuggable*. Since the proposed set of criteria addresses all the risks listed in the *OWASP Mobile Top 10* project, we can consider this set as comprehensive.

5. Case study

In this section, we present the methodology used to assess the security of Android mobile banking applications. First, we select seven mobile banking applications from the Canadian market. Second, we describe the environment set up for the evaluation. Finally, we discuss the obtained results.

5.1 *Dataset*

Our dataset consists of seven Android mobile banking applications from the Canadian market. We collected the applications on the Google Play Store during the month of March 2022. In Google Play Store, we listed the top free applications in the finance category. Then we filtered out Canadian mobile banking applications with over 10,000 downloads. The result is the seven mobile banking applications that make up our dataset. Table 3 presents a summary of the applications selected for our case study. We anonymized the banks whose applications were assessed.

Table 3. Dataset

Mobile Banking Application	Version	Number of Downloads
B1	4.21	1M+
B2	8.35.0	5M+
B3	8.35.1	1M+
B4	20.36.1	1M+
B5	5.32.1	1M+
B6	4.13.0	500k+
B7	17.1.24	10k+

5.2 *Assessment environment*

In this case study, we focus only on the criteria of the category data in transit. We believe this is the most important category because the criteria in this category affect both the mobile application and the server. Moreover, this category addresses the third risk of the OWASP Mobile Top 10, which is insecure communication. Finally, a hacker can easily intercept traffic on an insecure wireless communication channel.

To assess mobile banking applications against the criteria of the category data in transit, we adopt a dynamic analysis. Our methodology is inspired by the work of Bojjagani and Sastry [12].

Figure 1 shows the network topology for dynamic analysis. The mobile banking application is installed and running on the mobile device (i.e., Samsung Galaxy S10e with Android version 12). All traffic between the mobile device and the bank's server passes through the fake access point. This fake access point helps to simulate the man-in-the-middle attack. For this purpose, we use a personal computer with an Intel Core i5-7200U 2.50 GHz processor, 8 GB RAM and the Ubuntu 20.04.4 LTS 64-bit operating system. This personal computer is equipped with Nogotofail [33].

Nogotofail [33] is an open-source software developed by Google to evaluate network security issues. Through this software, we can perform man-in-the-middle attacks on connections between the mobile banking application and the bank's server.

Login to the selected mobile banking applications requires bank accounts with the respective banks. Since we do not have this facility, we just made login attempts to the various applications with invalid credentials. Then, the generated traffic is attacked with the Nogotofail software.

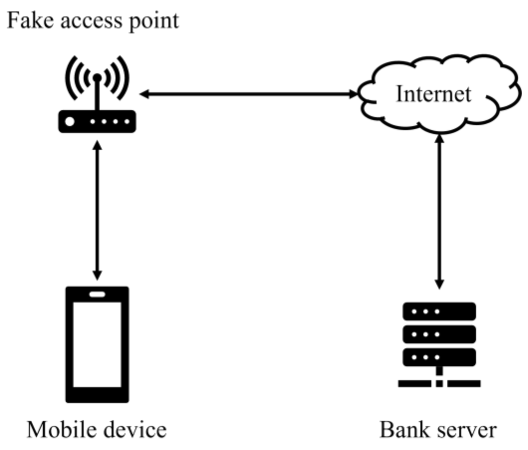

Figure 1. Network topology for dynamic analysis

5.3 *Findings*

Thanks to the Nogotofail tool, we were able to conduct a man-in-the-middle attack on the TLS connections initiated by the mobile banking applications. Table 4 presents the security assessment results.

Table 4. Security assessment results

Mobile Banking Application	Reject Self-Signed Certificate	Verify Certificate Hostname	Reject Expired or Revoked Certificate	Transmit Sensitive Data in Encrypted Form
B1	Yes	Yes	Yes	Yes
B2	Yes	Yes	Yes	Yes
B3	Yes	Yes	Yes	Yes
B4	Yes	Yes	Yes	Yes
B5	Yes	Yes	Yes	Yes
B6	Yes	Yes	Yes	Yes
B7	Yes	Yes	Yes	Yes

- Reject self-signed certificate. Instead of the real server certificate, we send to the mobile application a self-signed certificate by Nogotofail. All mobile banking applications rejected the self-signed certificate and aborted the connection.

- Verify certificate hostname. In this case, the attack consists of presenting the mobile banking application with a trusted certificate with a domain name other than the bank's server. For this purpose, we create a certificate authority (CA) that we add to our mobile device as a trusted CA. We then generate a trusted certificate with a different domain name than the bank's server. All mobile banking applications detected the invalid hostname and terminated the connection.

- Reject expired or revoked certificate. In this case, we present the mobile banking application with an expired certificate, but signed by a trusted CA. All mobile banking applications aborted the connection.

- Transmit sensitive data in encrypted form. Here, we inspect the traffic between the mobile application and the server for sensitive plaintext data. None of the mobile banking applications transmit sensitive data in plaintext.

All the assessed applications meet the security criteria of the category of data in transit. We can conclude that mobile banking applications in the Canadian market pay attention to the security of data in transit. However, the assessment conducted is partial since we are unable to log into the applications and attack the traffic generated by other features, such as balance inquiry and money transfer. This is a limitation of our work, since it is impossible to predict the outcome of attacks on traffic generated by other features (e.g., balance inquiry) of the tested mobile banking applications.

6. Conclusions

In this paper, we introduce a framework for assessing the security of Android mobile banking applications. This framework consists of a set of twenty-six criteria that mobile banking applications must meet to maintain an important level of security. We evaluated the proposed framework based on three requirements: *no redundancy*, *no ambiguity,* and *comprehensiveness*. Furthermore, we assessed the security of the mobile banking applications of seven major Canadian banks. This work can benefit both security researchers and Canadian banks. Security researchers can use this set of criteria for future security assessments of other mobile banking applications. In addition, they can design an automatic security assessment tool, taking into consideration the wide range of criteria proposed in this article. Canadian banks, in turn, can convince more customers to trust them, given the important level of security of their mobile banking applications. As a limitation, the security assessment of Android mobile banking applications carried out is partial. Indeed, we only carried out attacks on the traffic generated by the login operation to the applications with invalid credentials. Login to the various applications requires a bank account with the respective banks. Since we did not have this capability, we were unable to connect to the applications to carry out attacks on traffic generated by other features, such as money transfer and bill payment. Moreover, the proposed framework is dependent on the Android operating system. It does not consider vulnerabilities specific to other mobile operating systems, such as iOS. As future work, we can extend the proposed framework to cover vulnerabilities specific to the iOS operating system.

Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

Conflict of interest

There is no conflict of interest for this study.

References

[1] J. Chen, “Mobile Banking,” Accessed: Jul. 5, 2021. [online]. Available: https://www.investopedia.com/terms/m/mobile-banking.asp.

[2] “Focus: How Canadians Bank,” Accessed: Apr. 11, 2022. [online]. Available: https://cba.ca/technology-and-banking.

[3] E. Fernando, S. Surjandy, and M. Meyliana, “An Investigation Effective Factors Usage of Smartphone for Use Mobile Banking Services Case: Student University Customers,” in *Proc. 2020 Int. Conf. Inform. Manag. Technol. (ICIMTech)*, Bandung, Indonesia, Aug. 13–14, 2020, https://doi.org/10.1109/ICIMTech50083.2020.9211210.

[4] K. F. Arisya, Y. Ruldeviyani, R. Prakoso, and A. L. Fadhilah, “Measurement of Information Security Awareness Level: A Case Study of Mobile Banking (M-Banking) Users,” in *Proc. 2020 5th Int. Conf. Inform. Comput. (ICIC)*, Gorontalo, Indonesia, Nov. 3–4, 2020, https://doi.org/10.1109/ICIC50835.2020.9288516.

[5] “Cybersecurity in the Canadian Financial Sector as a National Economic Security Issue,” Accessed: Apr. 11, 2022. [online]. Available: https://www.ourcommons.ca/Content/Committee/421/SECU/Brief/BR10481660/br-external/InFidem-e.pdf.

[6] “Global market share held by mobile operating systems from 2009 to 2023, by quarter,” Accessed: Apr. 7, 2022. [online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[7] “About the OWASP Foundation,” Accessed: Sep. 29, 2021. [online]. Available: https://owasp.org/about/.

[8] “OWASP Mobile Top 10,” Accessed: Sep. 22, 2021. [online]. Available: https://owasp.org/www-project-mobile-top-10/.

[9] “App security best practices,” Accessed: Sep. 27, 2021. [online]. Available: https://developer.android.com/topic/security/best-practices.

[10] S. Chen, et al., “An empirical assessment of security risks of global android banking apps,” in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Seoul, South Korea, Jun. 27–Jul. 19, 2020, https://doi.org/10.1145/3377811.3380417.

[11] S. Chen, et al., “Are mobile banking apps secure? what can be improved?” in *Proc. 2018 26th ACM Joint Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Lake Buena Vista, FL, USA, Nov. 4–9, 2018, https://doi.org/10.1145/3236024.3275523.

[12] S. Bojjagani and V. V. Sastry, “VAPTAi: A Threat Model for Vulnerability Assessment and Penetration Testing of Android and iOS Mobile Banking Apps,” in *Proc. 2017 IEEE 3rd Int. Conf. Collaborat. Internet Comput. (CIC)*, San Jose, CA, USA, Oct. 15–17, 2017, https://doi.org/10.1109/CIC.2017.00022.

[13] X. Zheng, L. Pan, and E. Yilmaz, “Security analysis of modern mission critical android mobile applications,” in *Proc. ACSW 2017: Australasian Comput. Sci. Week 2017*, Geelong, Australia, Jan. 30–Feb. 3, 2017, https://doi.org/10.1145/3014812.3014814.

[14] R. Chanajitt, W. Viriyasitavat, and K.-K. R. Choo, “Forensic analysis and security assessment of Android m-banking apps,” *Aust. J. Forensic Sci.*, vol. 50, no. 1, pp. 3–19, 2016, https://doi.org/10.1080/00450618.2016.1182589.

[15] R. Kaur, Y. Li, J. Iqbal, H. Gonzalez, and N. Stakhanova, “A security assessment of HCE-NFC enabled e-wallet banking android apps,” in *Proc. 2018 IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Tokyo, Japan, Jul. 23–27, 2018, https://doi.org/10.1109/COMPSAC.2018.10282.

[16] S. Bojjagani and V. N. Sastry, “STAMBA: Security Testing for Android Mobile Banking Apps,” in *Advances in Signal Processing and Intelligent Recognition Systems*, Cham, Switzerland: Springer, https://doi.org/10.1007/978-3-319-28658-7_57.

[17] J.-H. Jung, J. Y. Kim, H.-C. Lee, and J. H. Yi, “Repackaging Attack on Android Banking Applications and Its Countermeasures,” *Wirel. Pers. Commun.*, vol. 73, no. 3, pp. 1421–1437, 2013, https://doi.org/10.1007/s11277-013-1258-x.

[18] B. Reaves, et al., “Mo (bile) money, mo (bile) problems: Analysis of branchless banking applications,” *ACM Trans. Priv. Secur.*, vol. 20, no. 1, pp. 1–31, 2017, https://doi.org/10.1145/3092368.

[19] D. Bassolé, G. Koala, Y. Traoré, and O. Sié, “Vulnerability Analysis in Mobile Banking and Payment Applications on Android in African Countries,” in *InterSol 2020: Innovations and Interdisciplinary Solutions for Underserved Areas*, Cham, Switzerland: Springer, https://doi.org/10.1007/978-3-030-51051-0_12.

[20] S. Castle, F. Pervaiz, G. Weld, F. Roesner, and R. Anderson, “Let’s talk money: Evaluating the security challenges of mobile money in the developing world,” in *Proc. 7th Annu. Symp. Comput. Develop.*, Nairobi, Kenya, Nov. 18–20, 2016, https://doi.org/10.1145/3001913.3001919.

[21] T. Chothia, F. D. Garcia, C. Heppell, and C. M. Stone, “Why Banker Bob (Still) Can’t Get TLS Right: A Security Analysis of TLS in Leading UK Banking Apps,” in *FC 2017: Financial Cryptography and Data Security*, Cham, Switzerland: Springer, 2017, https://doi.org/10.1007/978-3-319-70972-7_33.

[22] H. Darvish and M. Husain, “Security analysis of mobile money applications on android,” in *Proc. 2018 IEEE Int. Conf. Big Data (Big Data)*, Seattle, WA, USA, Dec. 10–13, 2018, https://doi.org/10.1109/BigData.2018.8622115.

[23] O. Osho, U. L. Mohammed, N. N. Nimzing, A. A. Uduimoh, and S. Misra, “Forensic Analysis of Mobile Banking Apps,” in *Computational Science and Its Applications—ICCSA 2019*, Cham, Switzerland: Springer, https://doi.org/10.1007/978-3-030-24308-1_49.

[24] A. A. Uduimoh, I. Ismaila, O. Osho, and S. I. M. Abdulhamid, “Forensic Analysis of Mobile Banking Applications In Nigeria,” *i-manager’s J. Mob. Appl. Technol.*, vol. 6, no. 1, pp. 9–20, 2019, https://doi.org/10.26634/jmt.6.1.15704.

[25] D. Wang and P. Wang, “Two Birds with One Stone: Two-Factor Authentication with Security Beyond Conventional Bound,” *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 4, pp. 708–722, 2016, https://doi.org/10.1109/tdsc.2016.2605087.

[26] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, “An empirical study of cryptographic misuse in android applications,” in *Proc. 2013 ACM SIGSAC Conf. Comput. Commun. Secur.*, Berlin, Germany, Nov. 4–8, 2013, https://doi.org/10.1145/2508859.2516693.

[27] M. Zarifopoulos and A. A. Economides, “Evaluating mobile banking portals,” *Int. J. Mob. Commun.*, vol. 7, no. 1, pp. 66–90, 2009, https://doi.org/10.1504/IJMC.2009.021673.

[28] “2021 CWE Top 25 Most Dangerous Software Weaknesses,” Accessed: Sep. 24, 2021. [online]. Available: https: //cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html.

[29] C. Thompson, R. Leininger, and R. Bhatt, “Mobile banking applications: security challenges for banks,” Accessed: Sep. 24, 2021. [online]. Available: https://www.accenture.com/_acnmedia/PDF-115/ Accenture-Mobile-Banking-Apps-Security-Challenges-Banks.pdf.

[30] M. Dworkin, “Recommendation for Block Cipher Modes of Operation Methods and Techniques,” NIST Special Publication 800-38A, 2001, https://doi.org/10.6028/NIST.SP.800-38A.

[31] M. S. Turan, E. B. Barker, W. E. Burr, and L. Chen, “Recommendation for Password-Based Key Derivation Part 1: Storage Applications,” NIST Special Publication 800-132, 2010, https://doi.org/10.6028/NIST.SP.800-132.

[32] “M5: Insufficient Cryptography,” Accessed: Oct. 11, 2021. [online]. Available: https://owasp.org/ www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography.

[33] “Google/Nogotofail (Version 1.2.0),” Accessed: Apr. 7, 2022. [online]. Available: https://github.com/google/ nogotofail.