Received: 7 February 2024 | Revised: 28 May 2024 | Accepted: 31 May 2024 | Published online: 19 June 2024

RESEARCH ARTICLE

Archives of Advanced Engineering Science
2024, Vol. 00(00) 1–11
DOI: 10.47852/bonviewAAES42022595

BON VIEW PUBLISHING

Analysis of Cybersecurity Vulnerabilities in Mobile Payment Applications

Esther Edem Archibong[1], Bliss Utibe-Abasi Stephen[1,*] and Philip Asuquo[1]

[1]*Computer Engineering Department, University of Uyo, Nigeria*

Abstract: Skepticism about the security of mobile payment applications has plagued user adoption of such platforms in some countries. Software developers have generally de-emphasized core principles guiding delivering safe mobile applications since for mobile payment applications, movement of monetary value is their priority. We find in surveyed literature that this situation is prevalent in low economy/low financial inclusion countries. Fifty fintech and traditional banks m-payment applications in both high and lower economic and technological advancement (high E&T apps and lower E&T apps, respectively) were selected in countries in Africa. This work may have significance in finance or economy, but it is mainly to unravel cybersecurity concerns. The analyses (static and dynamic) of the applications targeted the top 10 vulnerabilities on the 2023 Common Weakness Enumeration and Open Worldwide Application Security Project lists. The study employed Mobile Security Framework (MobSF) as the primary tool for both Android and iOS applications, while the Automated Security Risk Assessment tool was used to validate the vulnerabilities reported by MobSF. Results show that traditional m-payment apps were generally more secure than fintech m-payment apps. In the latter category, vulnerabilities under the information leakage and cryptography category were the most prevalent. On average, no marked difference was observed in security performance between high E&T apps and lower E&T apps. Incorrect default permission, cleartext storage of sensitive information, use of risky cryptographic algorithm, use of insufficiently random values, and information exposure were the most prevalent vulnerabilities. Conversely, insecure implementation of Secure Socket Shell (SSL) and trusting all certificates or accepting self-signed certificates had the fewest occurrences. Poor code quality was the highest source of security vulnerabilities in the study. The declining statistics of SMS leakage in recent studies were confirmed in this work. The most implemented security measure was certificate pinning for preventing or detecting man-in-the-middle attacks.

Keywords: vulnerability analysis, CWE, OWASP, cryptography, certificates, cybersecurity

1. Introduction

In recent years, developed countries have relied heavily on the near instantaneous movement of vast and small digital money for payment of goods and services irrespective of physical distance [1]. When investigated further, the state of the industry report on mobile money by GSMA [2] shows massive mobile payment adoption in Africa with 166 live services, 781 million registered accounts, $42.9 billion transaction volume, and $836.5 billion transaction value. Mobile payment applications, also written as mobile payment apps or m-payment apps, have been a significant way to achieving such level of commerce in underbanked regions in Africa [3]. Mobile banking applications for a while depended on physical access to banks which underbanked regions are not privy to. This has brought on a proliferation of branchless and financial technology (fintech) banks. Though it has improved access to banking services, there is still wide skepticism surrounding them.

Although massive adoption of different forms of mobile payment apps has been witnessed in both high-income and low/middle-income countries (LMICs), hundreds of barriers threaten the use of this payment method, especially in LMICs. These barriers include but are not limited to security factors, integrity, and perceived trust [4, 5]. Vulnerabilities in mobile payment applications are more dominant in third-world regions than in Europe, for instance [6]. Gao and Waechter [7] established as adoption determinants perceived trust, benefit, and convenience. Integrity in mobile banking implies that mobile banking firms observe specific rules [8], while perceived trust is the user's willingness to be vulnerable.

Perceived trust influences customer's choice to use mobile payment, but with perceived security or perceived risk as mediation [9, 10]. Perceived risk can be classified into perceived information risk and perceived financial risk [11], as well as perceived performance risk [12]. Perceived financial risk refers to users' beliefs, sentiments, and behaviors of the risk made up of an m-payment app [13, 14]. Perceived performance risk is the extent to which a user evaluates an m-payment app to have features or performance alien to what he needs. All these forms of perceived risk are basically rooted in insecurity in the mobile payment

*Corresponding author: Bliss Utibe-Abasi Stephen, Computer Engineering Department, University of Uyo, Nigeria. Email: blissstephen@uniuyo.edu.ng.

applications. The scenario is more concerning for African countries where some emerging financial policies (or poor implementation of same) mean that users are compelled to use mobile payment applications, regardless of the perceived risks. This presents a rise in the number of users exposed to the vulnerabilities in the system. Take Nigeria, for instance, where a scattergun approach to cashless policies saw a meteoric rise in number of mobile payment app users in 2023. It is such forced exposure to vulnerabilities that informed the choice of Africa as a case study.

From the background provided so far, the problem x-rayed may have massive economic significance. But it is in the first instance a cybersecurity (IT) problem. Lien et al. [15] highlighted security requirements that would provide safety for people using mobile banking to include confidentiality, integrity, availability, and authentication. Security practices ensuring confidentiality are carried out to enforce that only an authorized person accesses the right kind of data. The availability of the mobile banking system is also paramount. Apart from these, putting modalities in place to authenticate the user and secure financial data from being altered by unauthorized users is of significant concern. To guide these modalities, this paper assesses the vulnerabilities in the most used mobile payment applications in a third-world continent – Africa. Conventions are often used as guides for the assessment of vulnerabilities, for example, Common Weakness Evaluation/Open Worldwide Application Security Project (CWE/OWASP). CWE is a community-developed list of standard software and hardware weaknesses dating back to their first release in 2006, while OWASP is an open-source, nonprofit foundation that works to improve software security by testing and reporting known software vulnerabilities. Both CWE and OWASP routinely release a list of reported vulnerabilities and rank them.

In summary, this study makes the following major contributions:

a. To enable us to focus on the most prevalent cyber security vulnerabilities, the work adopts top vulnerabilities ranked by community-named conventions – CWE and OWASP 2023.
b. To accommodate a bulging number of fintech mobile apps in Africa, the study sample features a 26:24 mix of fintech and traditional mobile banking applications in Africa. The study then compares the performance of the two classes of m-payment apps.
c. We also present a study sample uniformly covering the five regions in Africa, taking into cognizance countries' high economic and technological advancements (high E&T) and lower in economic and technological advancements (lower E&T), and a performance comparison between high E&T and lower E&T.

The remainder of this paper is structured as follows: Section 2 presents the review of related literature, Section 3 provides the methodology adopted for the vulnerability assessment, and Section 4 gives the results. The paper does not end without a conclusion, as found in Section 5.

2. Literature Review

There have been some advancements in the assessment of vulnerabilities of m-payment apps. Speaking generally, the lack of security awareness among developers has been a major cause of vulnerabilities in mobile applications [16]. In terms of attacks, man-in-the-middle (MITM) attacks were major attacks found in Abdullah and Zeebaree [16] and Shahriar et al. [17]. The work in [18] identified as main vulnerabilities non-obfuscation of source code, external storage access, exportation of activities to other apps, logs information, use of object de-serialization. In Africa, the region of interest to us, outstanding statistics were SMS spoofing, server attacks, MITM attacks, and non-definition of privacy perceptions by users from external libraries tracking within a context [19]. In fact, the use of vulnerable third-party libraries in mobile app development is a major source of vulnerabilities in mobile applications as a whole [20]. In a later work by Bassolé et al. [21], most mobile payment applications in Africa have access to precise location and write information to the SD card (71.7%), allow contacts to be read (60.38%), and provide access to the camera (45.28%).

Approaches adopted for vulnerabilities assessment have predominantly been static and dynamic analyses [17, 22]. Static analysis (SA) generally encompasses scanning the source code or object code of an application and examining it without having to execute the program [23]. Dynamic analysis (DA) is used to detect vulnerabilities which occur during the run-time of an application cycle [18]. Unlike SA, DA is more complex and requires the installation of additional applications simulation of user input for analysis-based proposes.

Automated Security Risk Assessment (AUSERA), a system of security risk assessment automated on three levels, was applied in Chen et al. [6]. Vulnerabilities assessed in [6] include input harvest, capturing of application can input data that is sensitive like users' transaction information, data storage (considering if the application writes to external storage); data transmission (transferring data that is sensitive through SMS, data leakage); and communication structure. In the same work, AUSERA, although limited to Android, outperformed Qihoo360, AndroBugs, Mobile Security Framework (MobSF), and Quick Android Review Kit (QARK) in both precision and time cost [6].

Interestingly, prior works have mostly assessed/analyzed only Android permissions, with very few considering iOS permissions. Also, though fintech mobile payment applications have become nearly mainstream in African countries [22], they have not had the needed attention in prior vulnerability analyses. Researches classifying vulnerabilities in mobile payment applications according to community-named conventions are not prevalent yet – in the selected literature, such adoption was found only in Reaves et al. [24]. Adoption of such standards allows the focus of resources on statistically critical vulnerabilities. We also observed that top vulnerabilities on CWE and OWASP listings are usually a superset of top occurring vulnerabilities in existing literature. For instance, Shahriar et al. [17] found that most malicious mobile attacks exploit vulnerabilities such as sensitive data leakage, unsecured sensitive data storage, and transition of data. These vulnerabilities are part of the CWE and OWASP top 10 listings.

3. Research Methodology

3.1. System model

The system proposed in the work is shown in Figure 1, specifically the analyses. As would be discussed in details subsequently, what is core to the analyses is assessing the vulnerabilities of select m-payment apps (testing applications). Scripts were written to query the internet for m-payment apps meeting the study's inclusion criteria. This was to automate the process. The assessment results were retrieved and analyzed to extract data related to the study objectives. The vulnerabilities assessment was carried out in a five-step process, as shown in

Figure 1
System model

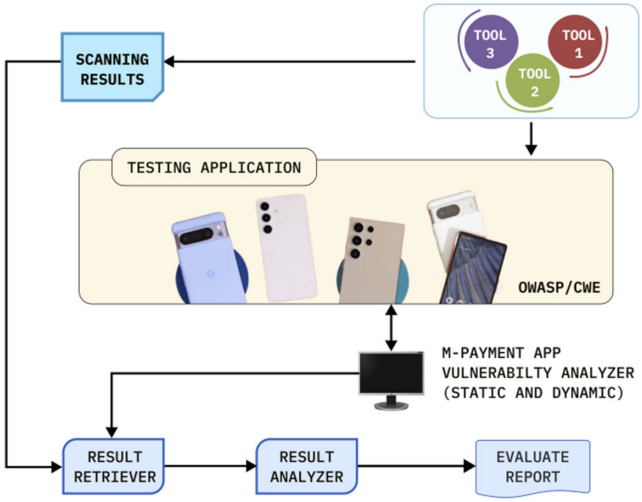
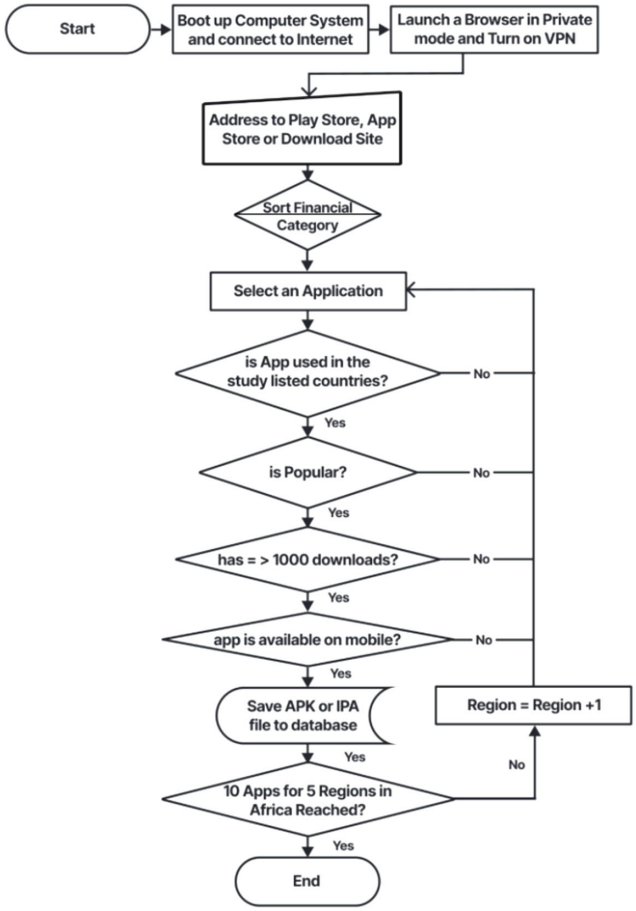


Figure 2
Research process flow

Figure 2, from region selection process, application selection process, static and dynamic analyses to result analysis.

Figure 3
Application selection process



3.2. Study region selection

First, low financial inclusion countries were identified, from which select mobile payment applications were used for the vulnerability assessment. The work focused on the five regions in Africa, namely, North Africa (Egypt and Morocco), Central Africa (Cameroon and Gabon), Eastern Africa (Rwanda and Kenya), Western Africa (Nigeria and Mali), and Southern Africa (Angola and South Africa). In selecting two countries from each region, different strategies were employed: first, population ratio, a country with a high population and another with less population for that region; second, economic and technological strengths, choosing a country with a good economy and technology, here on stated as high E&T (e.g., Egypt in the northern region, Cameroon in the central region, Rwanda in the eastern region, Nigeria in the western region, and South Africa in the southern region), and another with lower economic stability or technological advancement (lower E&T). This will enable the work to establish whether or not there is a relationship between economic and technology strengths of the countries considered and the security ratings of the apps in those countries.

3.3. App selection process

To select an application, considering the humongous number of applications available for mobile platforms, a sorting algorithm was employed. As depicted in Figure 3, the process starts by considering only applications found in the financial category of any of the Google Play Store and App Store. Next, it screens out applications not used in the countries of interest. Further, non-popular applications were screened out (applications below a 3-star rating). This enabled the selection of best-performing applications by user reviews. App ratings might not offer much insight to developers for improvements, but it sure shows how popular an application is with users [25]. To ensure applications considered for the work captures a fair population of users, anyone below 1,000 downloads was delisted. Third, a check for support on mobile operating systems was conducted to eliminate applications not available for mobile, considering the study is focused on payment applications that run on mobile devices. Further criteria considered were a non-duplication of any application (i.e., each application appears for only one country, with a total of five applications per country) and a ratio of traditional bank apps to fintech apps being 26:24. A total of fifty (50) applications were assessed.

3.4. Vulnerability analyses

3.4.1. Study taxonomy

The vulnerabilities assessed in the work followed the confidentiality, integrity, and availability (CIA) classification. Using the CWE/OWASP naming convention, confidentiality (certificates), integrity (information storage and cryptography), and availability (access control) vulnerability types were covered. We assessed the selected applications for the top ten (10) vulnerabilities in both CWE and OWASP listings for 2023 [26, 27]. Table 1 describes the study taxonomy listing different sensitive data that may be exposed to malicious entities,

Table 1
Study taxonomy by CWE/OWASP naming convention

ID	CWE/OWASP Top 10
Information/ storage leakage	
CWE-200	Exposure of sensitive information to an unauthorized actor
CWE-276/ OWASP: M2	Incorrect default permission/insecure data storage
CWE-312/ OWASP: M9	Cleartext storage of sensitive information/ reverse engineering
Access control	
CWE-749/ OWASP: M1	Exposed dangerous method or function/ improper platform usage
CWE-919	Weakness in mobile applications (WebView is enabled)
CWE-89/ OWASP: M7	Improper SQL element use (SQL injection)/ client code quality
Cryptography	
CWE-327/ OWASP: M5	Use a broken or unreliable cryptographic algorithm/insufficient cryptography
CWE-330	Use of insufficiently random values
CWE-649/ OWASP: M5	Dependence on obfuscation or encryption of security-connected inputs without checking integrity
Certificate	
CWE-295	Improper certificate validation

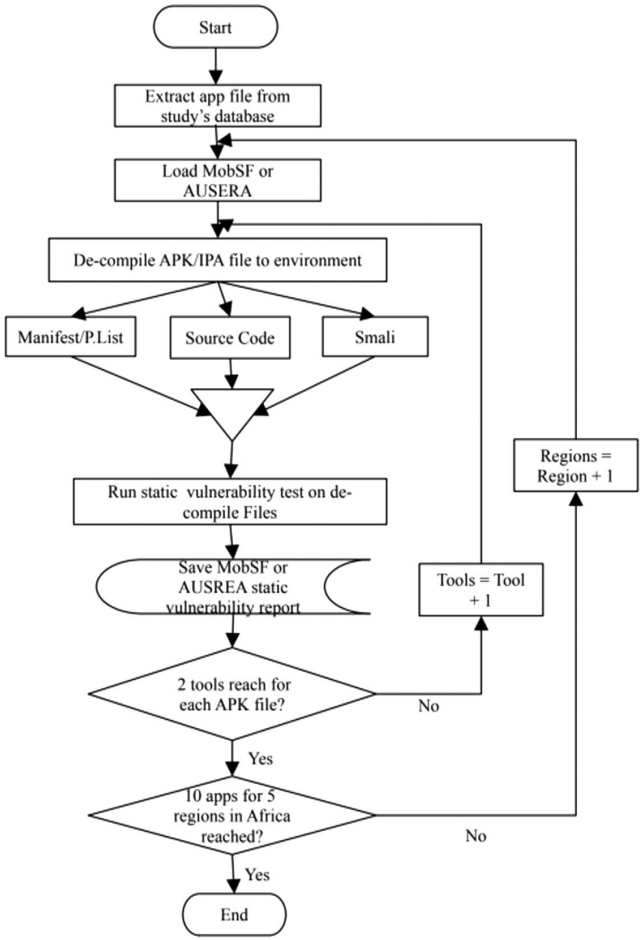
particularly CWE and OWASP top ten (10) vulnerabilities. Only 40% of the applications were checked for both iOS and Android permissions. The remaining 60% were assessed for just Android permissions.

3.4.2. Experiment setup

The system requirements were two personal computers, one running Linux Ubuntu 23.04 Lunar Lobster OS and the other an M1 chip Apple MacBook, and a 32-channel Mi-Fi. The tools included the Anaconda suite with Python 2.7, 3.6, and 3.8 loaded with MobSF, QARK, and AUSERA requirements libraries, Xcode, Fish terminal, Apktool version 2.3, MobSF framework, and AUSERA. MobSF as the primary tool was used for the study's SA, for both Android and iOS applications. AUSERA tool was used to validate the vulnerabilities reported by MobSF, thus reducing the false positives reported by each vulnerability analysis tool. However, due to the limitation of AUSERA, supporting only Android operating system applications, iOS applications SA was conducted on MobSF only.

a. *Static analysis:* To conduct the SA, as depicted in Figure 4, an application raw file saved to the study database from the application selection process is extracted. The MobSF environment is started via a terminal running on the local host using port 8080 to produce the MobSF graphic user interface (GUI). The extracted application file is uploaded to MobSF using the upload a file button. Using different plug-ins coded into MobSF, such as dex and smali, the uploaded file is decomposed into various files depending on the application type uploaded. On successful decomposition of application file to different file, analysis is automatically executed for each file, testing for vulnerabilities in application source code, application manifest/plist, dex class or method, network configuration, and certificate configuration and signature. This process is repeated until all applications for the study dataset are exhausted. The same procedures are employed for SA with AUSERA. However, AUSERA does not support GUI and iOS applications. Therefore, all analyses are carried out on a command line interface (CLI) terminal and for Android applications only.

Figure 4
Static analysis flow process



b. *Dynamic Analysis:* DA was implemented using the same tools and assessment settings as in SA. The process began with extraction of the application raw file saved to the study database during the application selection process. Next, an Android Virtual Device (AVD) emulator was launched, and the extracted raw file was installed and then launched on the virtual device. MobSF environment was started via a fish terminal running on the local host using port 8000 to produce MobSF GUI. MobSF is connected to the AVD to perform real-time vulnerabilities' detection on the running application.

4. Results

This section presents performance evaluation results and a discussion on the vulnerabilities and vulnerability sources observed in the study.

4.1. Performance evaluation

A security grading system was also used in evaluating performance of the apps, namely, A=70–100, B=60–69, C=50–59, D=45–49, E=40–44, and F=0–39. The security scores are shown at the foot of the heatmaps in Figures 6, 7, 8, 9, and 10. Scores with asterisks are from high E&T apps (Egypt in the northern region, Cameroon in the central, Rwanda in the Eastern, Nigeria in the western, and South Africa in the southern region). At the header of the heatmaps are the m-payment apps listed in blue for traditional banks and red for fintech banks. A vast majority (70%) of the applications performed at grade C level, as seen in Figure 5. A corroborating result can later be seen in Figure 10, where Mali in West Africa had the most secure applications, boasting one A-grade, three B-grade, and a C-grade payment application. Reaves et al. [24] study of seven mobile money wallets also achieved similar results for this region, reporting two A-grade applications in the form of Airtel mobile money in Western Africa and Zuum in the southern region. Figures 6, 7, 8, 9, and 10 provide details of the security scores of apps at the foot of the heatmaps. At the top, m-payment apps from traditional banks are listed in blue, while fintech ones are listed in red. Performance evaluation was done to confirm the impact of adopting the OWASP/CWE convention, compare the performance of traditional m-payment apps and those of fintech banks, and investigate if or not the economic and technological statuses of countries affect the security of m-payment apps in those countries.

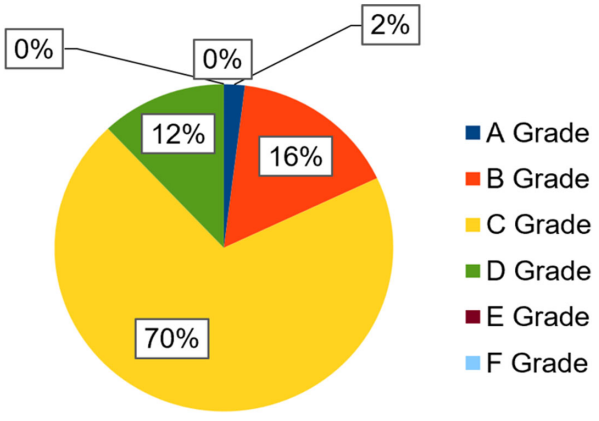
4.1.1. Adoption of OWASP/CWE convention

Figure 11 presents a justification for the use of OWASP/CWE top vulnerabilities in the analysis. In it, we have that 80% of vulnerabilities checked for had above 30% occurrences. Only two vulnerabilities (of the 10 adopted) turned out not to have significant occurrences. This is an improvement on study results where conventions like CWE and OWASP were not adopted as guides for top vulnerabilities. The use of CWE and OWASP listing helped the work target statistically critical vulnerabilities.

4.1.2. Traditional versus fintech apps

M-payment apps from traditional banks were found to be generally more secure than their fintech counterpart. Only in the western region did both classes have the same number of vulnerabilities. Specifically, as presented in Figures 6, 7, 8, 9, and 10, fintech apps generally failed assessments on information leakage and cryptography in the northern, central, and eastern regions.

Figure 5
Chart showing application vulnerability grades



4.1.3. Impact of economic and technological advantages

We observed a general marginal performance difference between m-payment apps in high E&T countries and those in lower E&T. We have 53.76 and 55.32 average security scores in high E&T apps and lower E&T apps, respectively. Similar marginality was observed when comparing the lowest security scores. However, in terms of highest security scores, northern and western regions had wide but converse margins in performance between high E&T apps and lower E&T apps. For instance, in the northern region, CIBW app (the best performing app in Egypt, a high E&T country) outperformed POKB (the best performing app in Morocco, a lower E&T country), by 10 points. Conversely, in the western region, it is the best performing mobile payment app (ATLQ) in a lower E&T country that outperforms its counterpart (GTWL) in the a high E&T country.

4.2. Vulnerabilities

4.2.1. CWE-200 (exposure of sensitive information to unauthorized actor)

We observed this vulnerability to be the fifth most occurring in the entire study, most dominant in m-payment apps in the northern and southern regions, and almost nonexistent in apps from the central region. Instances of disclosure of sensitive information were observed more in fintech apps than in m-payment apps from traditional banks. Specifically, violation of the principle of deny by default was observed.

4.2.2. CWE-276/OWASP: M2 (incorrect default permission/ insecure data storage)

Corroborating the study in Bassolé et al. [21], insecure data storage (CWE 276) was the highest occurring vulnerability, affecting 41 applications (82%) or at least seven applications from each region as seen in Figure 11. True Bill (TRBL) app's wrong implementation may store sensitive information in plaintext for JSON dump files. This vulnerability was also discovered in the L'BK app, which, on password reset, saved user details in cleartext. SGC's wrong implementation for TokenRequest led to user credentials being saved in cleartext. However, CashPlus Mobile Wallet (CPMW) stores user information temporarily in plaintext by permitting contents such as names to be copied to the clipboard. Like the CPMW app, the GLOP app creates a temporary file for writing session details such as authentication tokens. WHFF app exposes the user's IP address in plaintext using the verbose logging facility on HTTP error encounters. The full meaning of all mobile payment applications evaluated in this work is available in Figures 6, 7, 8, 9, and 10. The acronyms were used in the body of the work for easier reading.

4.2.3. CWE749/OWASP: M1 (exposed dangerous method or function/improper platform usage)

The exposure of a dangerous method or function (CWE 749) impacted 16 applications. This vulnerability implementation is closely related to CWE 919, in that it in most times involves JavaScript user code execution. Applications such as OME, OMPC, and UBAM were heavily affected by this vulnerability.

Figure 6

Vulnerabilities in Northern Africa m-payment apps. CIB Egypt (CIBE), CIB Smart Wallet (CIBW), L'bankalik (L'BK), Pocket Bank (POKB), CIH Mobile (CIHM), Orange Money Egypt(OME), Halan Lending (HLN), True Bill (TRBL), Waffarha (WFFH), CashPlus Mobile Wallet (CPMW)

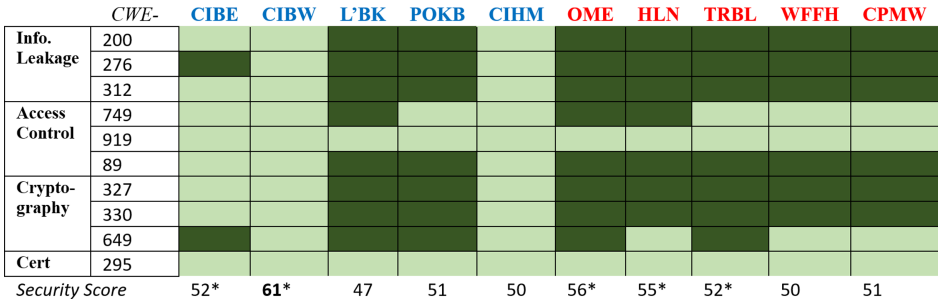


Figure 7

Vulnerabilities in Central Africa m-payment apps. NFC Bank (NFCB), AFG Mobile (AFGM), EBank Mobile (EBAM), GO2bank (GOBA), Mes Comptes-LCL (MESO), Orange Money Pro Cameroon (OMPC), SG Connect (SGC), Gabon Pay (GBPA), GamPay (GMPA), Glotelho Pay (GLOP)

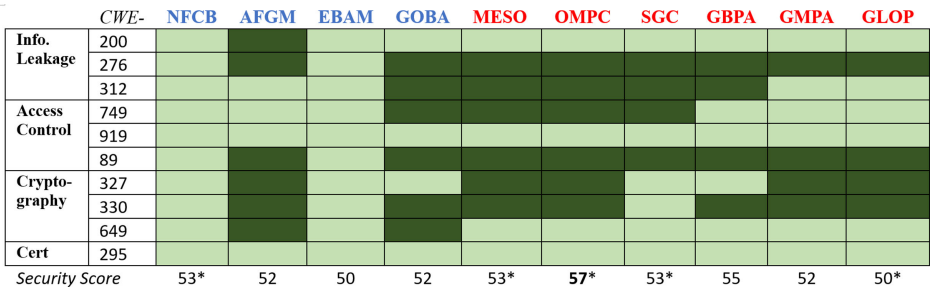
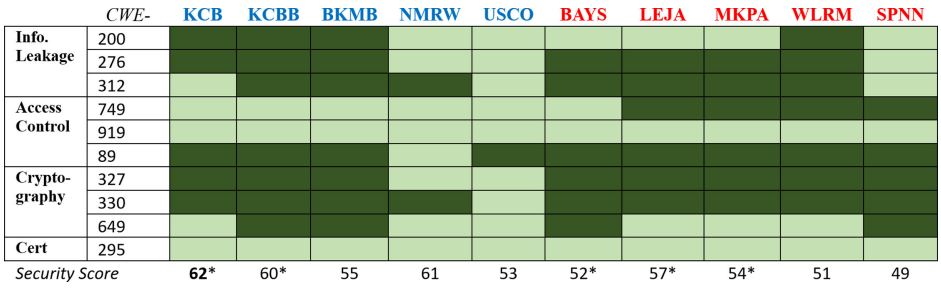


Figure 8

Vulnerabilities in Eastern Africa m-payment apps. KCB, KCB iBank (KCBB), BK Mobile (BKMB), NCBA Mobile Rwanda (NMRW), USACCO (USCO), Bayes (BAYS), Leja (LEJA), M-KOPA (MKPA), WorldRemit (WLRM), SPENN (SPNN)



4.2.4. CWE-919: weakness in mobile applications (WebView is enabled)

Access control such as WebView database view/debugging (CWE 919) is a critical security threat and should never be enabled. Within the study, this vulnerability was indicated as critical. However, as seen in Figure 11, only three applications (6%) were found to have implemented such a method. PLPY app in the western region of Africa, located in Nigeria, is the most affected. Static code analysis reveals multiple implementations of the application, giving access to remote WebView being enabled for in-app activities and HTTP errors (see Figure 8). In Figure 9, SA on the SPNN app reveals WebView debugging is enabled with an error message, creating an avenue for its user information being written in cleartext. BFFA app also implements this vulnerable method to manage its web activities. The vulnerability enables a malicious entity to be able to remotely view or change the internal set of the application, presenting such an entity with the power to effect memory modification to exploit the application.

4.2.5. CWE 89/OWASP: M7 (improper SQL element use (SQL injection)/(client code quality))

The majority of the applications when tested for improper SQL element use/SQL injection were found to utilize user input in back-

Figure 9
Vulnerabilities in Western Africa m-payment apps. Atlantique Mobile = ATLQ, MyBOA Mali = MBOA, UBA Mobile Banking (UBAM), First Bank Mobile (FRBM), GT World (GTWL), BIM Mobile Banking = BIMB, Coris Money = CORM, EBNDA = ENDA, OPAY, PalmPay (PLPY)

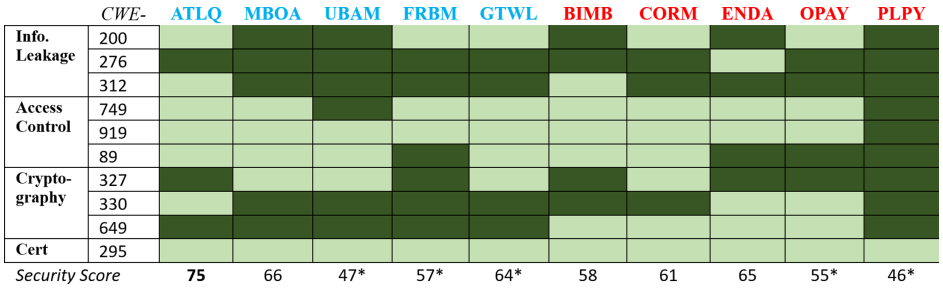
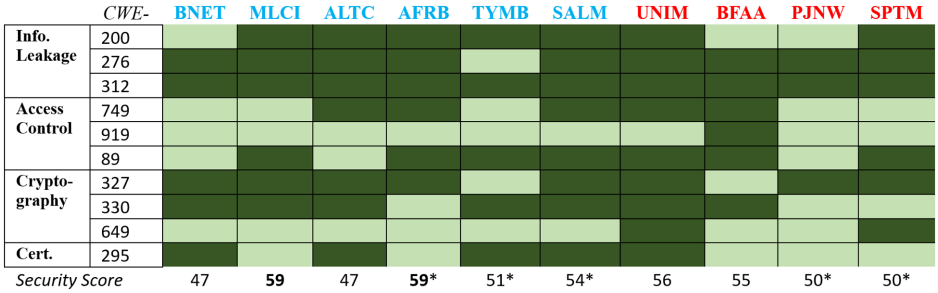


Figure 10
Vulnerabilities in Southern Africa m-payment apps. BCINET (BNET), Multicaixa (MLCI), Atlantico (ALTC), African Bank (AFRB), TymeBank (TYMB), SALAAM Africa Bank (SALM), UNITEL Money Parceiros (UNIM), BFA App (BFAA), PayJustNow (PJNW), Spot Money (SPTM)



end queries or commands. However, by injecting meta-characters, a malicious entity can execute malicious code that inadvertently will be interpreted as part of the command or query, whereby being able to retrieve arbitrary database records or manipulate the content of the back-end database. A total of 18 of the 50 applications tested employed this insecure method, such as BFAA, USCO, and MKPA apps.

4.2.6. CWE-312/OWASP: M9 (cleartext storage of sensitive information/reverse engineering)

Several applications analyzed were vulnerable to one or more wrong implementations of processing/storing user information with little or no encryption, exposing personal user information and data critical to transactional integrity through one of these methods, namely, enabling cleartext in the manifest to all or specific domains in scope, writing sensitive data to external storage, or user data logging. SA by MobSF showed 37 applications from the dataset store cleartext of sensitive information.

4.2.7. CWE 327/OWASP: M5 (use of broken or risky cryptographic algorithm/insufficient cryptography)

The use of a broken or risky cryptographic algorithm or insufficient cryptography affected 37 of the 50 applications analyzed, making it the second most prevalent vulnerability found in the work. It affected on average seven applications from each region. However, the majority of these implementations were noncritical to the application functionalities. Nonetheless, the use of broken or risky cryptographic algorithms exposes users' sensitive information to malicious entities. The applications studied mostly used either SHA1 or MD5 to encrypt and decrypt the application's public and private key and signature during communication and data exchange.

4.2.8. CWE 330 (use of insufficient random values)

The use of insufficiently random values was found in 34 applications of the study's dataset. Due to the deterministic nature of computers or mobile devices, producing truly random numbers is fundamentally impossible. Pseudorandom number generators (RNG) are used to tackle this flaw; however, the quality of numbers generated varies with the type of RNG algorithm used and greatly impacts the degree of randomness resilient against prediction attacks. BKMB app utilizes weak default Java method for random number generator for generating its biometric data. GTWL app also implements a vulnerable default Java method for random number generation for its transaction requests_id. The application fails to generate a long random key for input; instead, 17 bits of key are generated randomly, and the remaining bits are padded with mobile number or date by the mobile operating system. WHFF app in contrast to the others is affected critically as it implements an insufficient random number generator paired with a weak hash algorithm (MD5) for validating communication handshake.

4.2.9. CWE-649/OWASP: M5 (insertion of sensitive info into log file)

A total of 18 applications (36%) were vulnerable to reliance on code obfuscation as seen in Figure 11. 100% of these applications

Figure 11
Percentage of apps per vulnerability

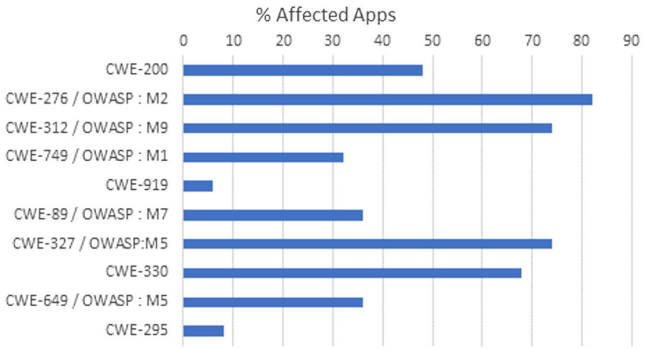
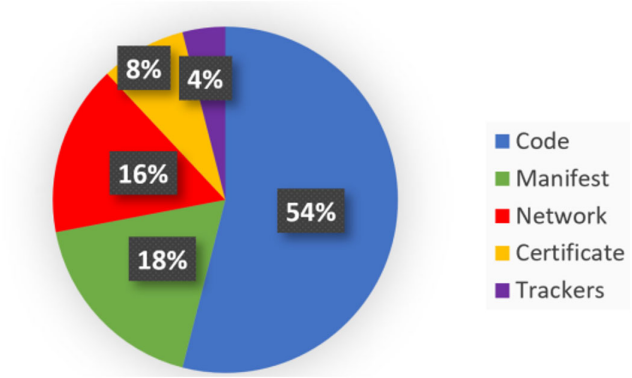


Figure 12
Distribution of vulnerability sources



failed the obfuscation test to protect its code structure. Using QARK to attempt reverse engineering in each application's APK yielded an 80% success rate for creating a malicious APK file using the vulnerabilities found within each application, given time and resources. Chen et al. [28] reportedly employed QARK for vulnerability fixing in their study. Nonetheless, they reported the tool accuracy inferior to other tools implemented within the study, suggesting the QARK exploitation function for creating vulnerable APK as its significant strength, not for performing analysis. However, QARK only supports Android application reverse engineering; the results obtained from this study, which exploited APK creation, remain inconclusive whether the same results can be obtained for the iOS application.

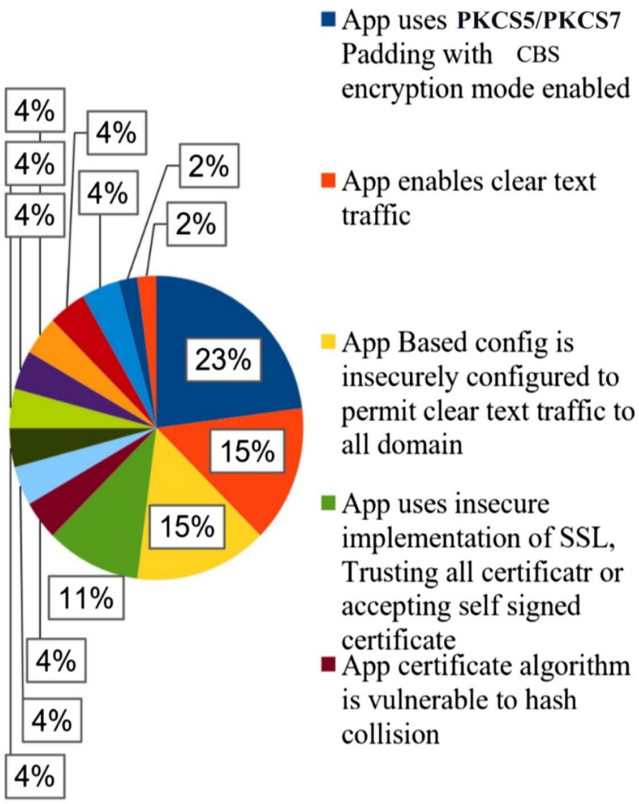
4.2.10. CWE-295 (improper certificate validation)

Improper validation of certificates remains a critical security loophole, and as shown by different works, many applications fail to validate Secure Socket Shell (SSL)/transport layer security (TLS) certificates properly. Applications that implement improper certificate validation are caused by poorly designed application programming interface (API) calls that make it easy to make validation mistakes. Four (4) applications within the study were found to have this vulnerability. Due to the severity of security this poses, it causes a severe threat to these application's functionality. The simple explanation for improper certificate validation implementation may present itself as, during the development phase, developers often opt in for more straightforward but less secure solutions such as accepting any certificate or self-signed certificate to speed up the development phase. However, these less secure solutions are often deployed to application production code, exposing the application to MITM attacks. Two key issues that should be addressed to mitigate this practice include the following:

a. Verifying a certificate comes from a trusted source such as certificate authority, and determining the endpoint server presents the proper certificate before authentication. In the case of the ALTCO app, which fails to conduct valid hostname verification, this results from failing to follow the recommended Hostname Verifier method recommended in the official Android and iOS documentation.
b. When there is a lack of proper certificate management practices implemented at the application layer (code), the most viable solutions are to enforce SSL/TLS configuration at the operating system layer and certificate pinning in both code and application manifest. This solution has been adopted, as shown in Figure 12, with 54% of the applications implementing certificate pinning to detect or prevent MITM attacks. Two applications implement a certificate that does not expire to caution certificate date errors. Fahl et al. [29] demonstrated this method of securing an application in the event of trusted root certificate compromise, which entails employing DV Cert pinning to protect against MITM.

As seen in Figure 13, Public Key Cryptography Standard #5/#7 (PKCS5/PKCS7) padding with Cipher-Block-Chaining (CBC) mode enabled is the most frequent (23%) cryptographic vulnerability within the study, affecting 11 applications critically. The difference between the two padding mechanisms lies in block

Figure 13
Chart showing occurrence level of the vulnerabilities



size; PKCS5 padding is limited to 8-byte block sizes, and PKCS7 padding works for block sizes from 1 to 255 bytes. In the past, this cryptography algorithm was considered secure. However, the Microsoft Vulnerabilities Report for 2023 captured that this method is no longer secure to decrypt data encrypted with the CBC mode of symmetric encryption [30]. It further shows that the method is vulnerable to constant timing attacks, which rely on the ability to change the encrypted data and test the result with the oracle due to timing differences. To mitigate this vulnerability, all applications implementing the PKCS5/PKCS7 algorithm, such as the AFRB app, should enforce an encrypt-then-sign model, that is, should create a signature for its data and validate the created signature before any data exchange or other operations are performed. Data integrity checks such as keyed-hash message authentication code (HMAC), which validate at constant time comparison before decrypting the data, should be paired with PKCS5/PKCS7 padding and implemented in check before the decrypt method.

4.2.11. Other vulnerabilities

The use of four or five-digit personal identification numbers (PINs) was the default authentication implementation for the majority of applications in the dynamic analysis section of the study. PINs are weak against brute force attacks, as Chanda [31] and Guerar et al. [32] suggested that a solid password provides better security to applications when compared to PINs. L'BK app was found to save password reset in plaintext. This does not occur for the original set password. However, although this is not tested in a dynamic environment and may be a false positive by MobSF, the code structure in three different locations suggests differences.

Six applications were found to have wrongly implemented code, which exposes the application to this vulnerability. Of the six applications, five were located in the northern region of the continent, suggesting variation in factors influencing such attributes.

Permission to read and write sensitive and nonsensitive application data to external storage such as SD disk remained high among applications within the study regardless of numerous efforts from Android, iOS, and other study documentation warning of the critical security implication this practice leads to. On average, 80% of the study's dataset applications enabled permission to read and write data for both operating systems.

Applications such as the UBAM app enable user data backup to external storage. On further DA, the backup data contains a cleartext of user-sensitive information. In addition to cleartext being contained in the backup data, the backup data was recognized by the UBAM app when uploaded to a different device. However, given that the UBAM app performs a device connected to an account check, this weakness may not affect the application critically.

Camera access/screenshot permission was enabled for 70% of the applications from different regions except for the southern region, which had all its applications enable permission for camera access/screenshot. This permission evades the application user privacy as it can capture activities by taking screenshots of transactions or take photography using the primary or secondary camera. Similar results for input harvest were obtained in Chen et al. [28]. Of the 50 applications analyzed, only the GTWL app application was protected from screenshots of critical screens such as transaction history and account balance. This vulnerability is primarily caused by developers failing to implement anti-screenshot harvest code, such as setting the isScreenCaptureEnabled flag to disable. Only five (10%) of the applications enabled permission for SMS access, agreeing with findings in Chen et al. [6], and a significant improvement on what it was in Castle et al. [19], where SMS spoofing was found to be the most significant threat.

4.3 Vulnerability sources

4.3.1. Code

Vulnerabilities arising from wrong implementation code practice and code quality were the highest source of security vulnerabilities, affecting 27 applications (54%) from the study dataset (see Figure 13). Wrong implementation of PKCS5/PKCS7 with CBC mode enabled without data integrity check code affected 11 applications, critically rendering these applications vulnerable to oracle padding attack; implementing an insecure random number generator critically affected two applications due to poor code implementation.

4.3.2. Manifests

Vulnerabilities resulting from the wrong configuration of the manifest file were the second most notable cause of security vulnerability, affecting nine applications (18%), as seen in Figure 13. Of the nine applications affected, seven applications enabled cleartext traffic, while two applications enabled application data backup in their manifest. As Chen et al. [33] concluded, such practice creates room for user information leakage. The insecure configuration of network protocol critically affected eight applications; seven had its base configuration insecurely configured to permit cleartext traffic to all domains defined in its application scope. Sivakorn et al. [34] showed in their study that implementing such an insecure domain configuration for cleartext leads to sensitive user information being exposed to malicious entities, even with HTTPS protocols are enabled.

4.3.3. Network

Proper network configuration strengthened the security of five applications. These applications enabled network-based configuration files to disallow cleartext traffic to all domains, forcing all communication to be encrypted appropriately and transmitted using SSL.

4.3.4. Trackers

Accessing coarse location (GPS) and fine location (network location) was enabled for 60% of the applications. However, this privacy vulnerability was more significant in the western region, affecting 90% of applications for fine location and 80% for coarse location. These applications can track the user's location. Contrary to popular belief that iOS manages user tracking permission better than Android, the study found that 100% of applications that enabled Android tracking permission did the same for iOS. Nevertheless, iOS enabled this permission when needed, not at install time. Moreover, iOS indicated whenever an application tracking was in use. Therefore, iOS did notify its users when an application was tracking their location. Kollnig et al. [35] and Yin [36] in their studies obtained similar results, concluding that although users' perceived trust for privacy within the iOS ecosystem was high, in reality, applications in the Android ecosystem performed similar location data requests as those for the iOS.

4.3.5. Certificates

Poor code quality was the leading cause of critical security concerns, and properly implemented source code was also determined to be the most significant source of security strength for most applications. Twenty-seven (27) applications of the study dataset implemented SSL certificate pinning through proper coding to detect MITM attacks, while two applications implemented code to tackle anti-tap jacking attacks.

5. Conclusion

We have had vulnerabilities in mobile payment applications in third-world countries pushing people from mobile payment applications, on one hand. On the other hand, we have had fast-paced growth of smartphones and the widespread of fintech organizations attracting users to this payment method for daily payment of goods and services. The fact that the later factor is having superior influence means there is a net pull to adoption of mobile payment applications. Supply meeting demand, banks find it profitable to roll out these applications. This scenario means the vulnerabilities affect many users rather than a few. This creates the need to analyze how secure these mobile payment applications used daily for different transactions.

Case study applications, countries, and tools were carefully selected using a sorting algorithm to produce the best possible representatives for each case respectively. Fifty applications were selected from five African regions, choosing two countries from each region and five payment applications from each country. Furthermore, the study taxonomy was carefully selected to best present top vulnerabilities specific to mobile payment applications. These were categorized into information/storage leakage, access control, cryptography, and certificate vulnerabilities, with a couple of specific vulnerabilities in each category. Using the top 10 vulnerabilities on CWE and OWASP listing proved significant, as a vast majority of vulnerabilities checked for were statistically critical.

The study uses a hybrid vulnerability analysis method, employing SA for source code, meta-file, and privacy analysis. DA was used for application behavior, connections, and authentication analysis. Tools used for SA included MobSF and AUSERA, while MobSF was used primarily for DA. QARK was used for testing applications that showed weakness in proper code obfuscation during SA.

A vast majority of mobile payment applications had average overall performance on the assessments conducted.

Recommendations

Failure to properly analyze the weaknesses and threat channels to an application will negatively impact the revenue, trust, integrity, and reliance of any financial payment institute. Therefore, application packing protection methods for payment applications are necessary. Financial entities should implement protection file packages such as APKProtect or Bangcle, as this increases the exploitation difficulty. Additionally, integration of third-party libraries remains the easiest way of introducing weaknesses for one's application; hence, a careful selection process should be implemented when choosing third-party libraries to manage risk effectively.

Funding Support

This work is sponsored by 2019 Project of Humanities and Social Sciences of Henan Provincial Department of Education: "Research on Translation for Overseas Publicity from the Perspective of the Persuasion Theory in Western Rhetoric" (2019-ZZJH-643); 2019 Teaching Reform Project of School of Foreign Studies, North China University of Water Resources and Electric Power: "The Reforming Design and Practice of Mixed Teaching of Online and Offline Course for English Rhetoric"; and 2019 Teaching Reform Project of Henan Province: A Research on the "Golden Lesson" of College English from the Perspective of Telling Chinese Stories (2019SJGLX284).

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data available on request from the corresponding author upon reasonable request.

Author Contribution Statement

Esther Edem Archibong: Methodology, Software, Investigation, Resources, Visualization. **Bliss Utibe-Abasi Stephen:** Validation, Data curation, Writing – original draft, Supervision, Project administration. **Philip Asuquo:** Conceptualization, Formal analysis, Writing – review & editing.

References

[1] Bech, M. L., Faruqui, U., & Shirakami, T. (2020). Payments without borders. *International Journal of Computer Science and Information Security*, *14*(7), 77–78.

[2] GSMA. (2023). *State of the industry report on mobile money 2023*. Retrieved from: https://www.gsma.com/sotir/wp-ontent/uploads/2023/04/GSMA-State_of_the_Industry-2023_Exce-summary-ENG.pdf

[3] Mishra, V., Walsh, I., & Srivastava, A. (2022). Merchants' adoption of mobile payment in emerging economies: The case of unorganised retailers in India. *European Journal of Information Systems*, *31*(1), 74–90.

[4] Dinh, V. S., Nguyen, H. V., & Nguyen, T. N. (2018). Cash or cashless? Promoting consumers' adoption of mobile payments in an emerging economy. *Strategic Direction*, *34*(1), 1–4.

[5] Cham, T. H., Cheah, J. H., Cheng, B. L., & Lim, X. J. (2021). I am too old for this! Barriers contributing to the non-adoption of mobile payment. *International Journal of Bank Marketing*, *40*(5), 1017–1050.

[6] Chen, S., Zhang, Y., Fan, L., Li, J., & Liu, Y. (2023). Automated security vulnerability detection for android apps. *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, *154*, 1–5.

[7] Gao, L., & Waechter, K. A. (2017). Examining the role of initial trust in user adoption of mobile payment services: An empirical investigation. *Information Systems Frontiers*, *19*(3), 525–548.

[8] Khan, M. R., Rana, S., & Hosen, M. I. (2022). Impact of trustworthiness on the usage of M-banking apps: A study on Bangladeshi consumers. *Business Perspectives and Research*, *10*(2), 234–250.

[9] O'kello, G., & Munene, J. C. (2021). Analyzing the relationship between mobile money adoption and usage and financial inclusion of MSMEs in developing countries: Mediating role of cultural norms in Uganda. *Journal of African Business*, *22*(1), 1–20.

[10] Sharma, S. (2023). A study of vulnerability scanners for detecting SQL injection and XSS attack in websites. *Artificial Intelligence and Applications*, *1*(4), 214–220.

[11] Ma, L., Su, X., Yu, Y., Wang, C., Lin, K., & Lin, M. (2018). What drives the use of m-payment? An empirical study about Alipay and WeChat payment. In *2018 15th International Conference on Service Systems and Service Management (ICSSSM)*, *2018*, 1–6.

[12] Chang, W., Chen, L. M., & Hashimoto, T. (2021). Cashless Japan: Unlocking influential risk on mobile payment service, *Information Systems Frontiers*, *24*(5), 1515–1528.

[13] Nguyen, L., Gallery, G., & Newton, C. (2019). The joint influence of financial risk perception and risk tolerance on individual investment decision-making. *Accounting & Finance*, *59*, 747–771.

[14] Asuquo, P., Usoh, M., Stephen, B., Chikezie Samuel, A., & Awodeyi, A. (2022). Cyber-physical systems attacks and countermeasures. In J. S. Banerjee, S. Bhattacharyya, A. J. Obaid & W. C. Yeh (Eds.), *Intelligent cyber-physical systems security for industry 4.0* (pp. 119–145). Chapman and Hall/CRC.

[15] Lien, J., Hughes, L., Kina, J., & Villasenor, J. (2015). Mobile money solutions for a smartphone-dominated world. *Journal of Payments Strategy and Systems*, *9*(3), 341–350.

[16] Abdullah, H., & Zeebaree, S. R. (2021). Android mobile applications vulnerabilities and prevention methods: A review. In *2021 2nd Information Technology to Enhance e-learning and Other Application*, 148–153.

[17] Shahriar, H., Zhang, C., Talukder, M. A., & Islam, S. (2021). Mobile application security using static and dynamic analysis. In Y. Maleh, M. Shojafar, M. Alazab & Y. Baddi (Eds.), *Machine intelligence and big data analytics for cybersecurity applications* (pp. 443–459). Springer Cham.

[18] Bojjagani, S., & Sastry, V. N. (2017). VAPTAi: A threat model for vulnerability assessment and penetration testing of Android and iOS mobile banking apps. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing*, (pp. 77–86).

[19] Castle, S., Pervaiz, F., Weld, G., Roesner, F., & Anderson, R. (2016). Let's talk money: evaluating the security challenges of mobile money in the developing world. *Proceedings of the 7th Annual Symposium on Computing for Development*, *2016*, 1–10.

[20] Zhan, X., Fan, L., Chen, S., Wu, F., Liu, T., Luo, X., & Liu, Y. (2021). ATVHunter: Reliable version detection of third-party libraries for vulnerability identification in Android applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*, 1695–1707.

[21] Bassolé, D., Koala, G., Traoré, Y., & Sié, O. (2020). Vulnerability analysis in mobile banking and payment applications on Android in African countries. In J. P. R. Thorn, A. Gueye & A. P. Hejnowicz (Eds.), *Innovations and interdisciplinary solutions for underserved areas: 4th EAI international conference, InterSol 2020, Nairobi, Kenya* (pp. 164–175). Springer International Publishing.

[22] Bech, M. L., & Hancock, J. (2020). Innovations in payments. *Journal of Computer and Information*, *7*(6), 11–25.

[23] Yang, T., Yang, Y., Qian, K., Lo, D. C.-T., Qian, Y., & Tao, L. (2015). Automated detection and analysis for Android ransomware. In *17th International Conference on High Performance Computing and Communications; 7th International Symposium on Cyberspace Safety and Security, and 12th International Conference on Embedded Software and Systems*, 1338–1343.

[24] Reaves, B., Bowers, J., Scaife, N., Bates, A., Bhartiya, A., Traynor, P., & Butler, K. R. (2017). Mo(bile) money, mo(bile) problems: Analysis of branchless banking applications. *ACM Transactions on Privacy and Security*, *20*(3), 1–31.

[25] Huebner, J., Schmid, C., Bouguerra, M., & Ilic, A. (2019). FinMARS: A mobile app rating scale for finance apps. In *Proceedings of the 9th International Conference on Information Communication and Management*, 6–11.

[26] OWASP. (2023). *OWASP API security top 10 2023 has been released | OWASP foundation*. OWASP. Retrieved from: https://owasp.org/blog/2023/07/03/owasp-api-top10-2023.html

[27] OWASP. (2023). *OWASP top ten | OWASP foundation*. Retrieved from: https://owasp.org/www-project-top-ten

[28] Chen, S., Su, T., Fan, L., Meng, G., Xue, M., Liu, Y., & Xu, L. (2018). Are mobile banking apps secure? What can be improved? *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, *2018*, 797–802.

[29] Fahl, S., Harbach, M., Perl, H., Koetter, M., & Smith, M. (2013). Rethinking SSL development in an appified world. *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, 39–50.

[30] Microsoft. (2023). *Microsoft vulnerabilities report 2023*. Retrieved from: https://www.beyondtrust.com/resources/whitepapers/microsoft-vulnerability-report

[31] Chanda, K. (2016). Password security: An analysis of password strengths and vulnerabilities. *International Journal of Computer Network and Information Security*, *8*(7), 23–30.

[32] Guerar, M., Migliardi, M., Palmieri, F., Verderame, L., & Merlo, A. (2020). Securing PIN-based authentication in smartwatches with just two gestures. *Concurrency and Computation: Practice and Experience*, *32*(18), 46–56.

[33] Chen, S., Fan, L., Meng, G., Su, T., Xue, M., Xue, Y., . . . , Xu, L. (2020). An empirical assessment of security risks of global Android banking apps. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, *2020*, 1310–1322.

[34] Sivakorn, S., Keromytis, A. D., & Polakis, J. (2016). That's the way the cookie crumbles: Evaluating https enforcing mechanisms. *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, Malaysia*, *2016*, 71–81.

[35] Kollnig, K., Shuba, A., Binns, R., Van Kleek, M., & Shadbolt, N. (2022). Are iPhones really better for privacy? Comparative study of iOS and Android apps. *Proceedings on Privacy Enhancing Technologies*, *22*(2), 6–24.

[36] Yin, Z. (2017). Dynamic analysis methods of iOS application security. *DEStech Transactions on Computer Science and Engineering*, *19*(4), 372–385.

How to Cite: Archibong, E. E., Stephen, B. U.-A., & Asuquo, P. (2024). Analysis of Cybersecurity Vulnerabilities in Mobile Payment Applications. *Archives of Advanced Engineering Science*. https://doi.org/10.47852/bonviewAAES42022595