2017

OpenSSL vulnerabilities in mobile banking applications

Paul William Muriuki
Faculty of Information Technology (FIT)
Strathmore University

OPENSSL VULNERABILITIES IN MOBILE BANKING APPLICATIONS

PAUL WILLIAM MURIUKI

89707

Submitted in partial fulfilment towards the study for a Master's Degree in Information Systems Security at Strathmore University

Faculty of Information Technology

Strathmore University

Nairobi, Kenya

March 2017

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Paul William Muriuki

…….

28th June, 2017

Approval

The thesis of Paul William Muriuki was reviewed and approved by the following:

Dr. Matunda Nyanchama

Faculty of Information Technology

Strathmore University

Dr. Joseph Orero,

Dean, Faculty of Information Technology

Strathmore University

Professor Ruth Kiraka,

Dean, School of Graduate Studies,

Strathmore University

Abstract

Mobile banking has taken Kenya by storm. It is an easy and convenient banking channel in your hands. It is accessible from anywhere provided you have an internet connection or connectivity to your mobile network provider. Banks and other financial institutions have seen the numerous benefits of providing mobile banking services to their customers and each one is rushing to deploy their own mobile banking solution in an attempt to gain a competitive edge over their competitors.

But as with new inventions, particularly those aimed at people's finances, existing and potential clients of these institutions are worried as to how safe their transactions are. This is especially so since they are effected from remote locations and through their mobile devices. Questions are being asked concerning how secure mobile banking is and how safe personal information is while being transmitted from mobile devices to banks.

This study seeks to understand the architecture of mobile banking solutions and identify potential areas of vulnerability in the systems deployed. It further seeks to look at how secure the deployed OpenSSL third party libraries are. Third party OpenSSL libraries are used extensively to secure data in transmission. The study, by leveraging the software development life cycle's Agile development methodology, proposes to provide a tool that can be used by financial institutions to test banking applications developed for mobile devices before deploying them to the market. This will ensure that only secure systems are deployed. The results of this study will show the importance of proper testing before application deployment.

Keywords: Mobile banking, mobile applications, apps, OpenSSL, third party libraries, financial institutions, financial services, mobile devices, Android.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to acknowledge those few who enabled me to successfully complete this research project.

I thank God for the guidance, encouragement and grace He has shown me throughout this course up until its completion with this dissertation.

I thank my supervisor, Dr. Matunda Nyanchama for the guidance and direction he has provided me in the course of this research.

I thank Collins Oduor for the continued support and guidance he has provided me as well.

Chapter 1: Introduction

1.1 Mobile Banking

Kenya is currently seen as the hotbed of mobile money innovations globally with the advent of platforms like M-PESA. Financial institutions in Kenya such as Equity Bank, Family Bank and Co-operative Bank have been able to leverage mobile application technology to enable them to directly provide financial services through mobile banking solutions such as Eazzy Banking, Pesa Pap and MCoop Cash respectively. This is an attempt at taking advantage of the technology and convenience that comes with mobile phones. (Serianu, 2015) has noted that previously unbanked people who could not be reached by traditional retail banking can now access mobile banking services due to the ubiquity of mobile devices. From various household surveys conducted (Porteous, 2006) in selected countries including Kenya, it is evident that within a decade or less of the rollout of mobile devices, as many, if not more, people have these devices as they do bank accounts even though the latter has been around longer than the former. This means that more banking customers can gain access to banking services through their mobile devices.

Mobile banking is a quick, simple and convenient way of taking command of one's bank account from one's mobile phone. It has enabled people to perform various banking transactions at their own convenience wherever they may be, provided there is decent network coverage. Mobile banking promises and delivers on a convenient way of accessing your bank services, including paying utility bills and making use of other available banking services. Some of these transaction services include, but are not limited to, performing balance and mini-statement requests, account to account deposits, purchasing of phone credit, loan application and even activating/deactivating credit/debit cards.

Forward thinking financial institutions are constantly looking for ways to leverage mobile technology in order to get the technological advantage over their competitors. For example, Equity bank has gone ahead a step and availed an Application Programming Interface (API) to mobile application developers in a bid to further development of innovations in Agency Banking, Merchant Banking and Mobile banking ("Equity bank's APIs are now open to developers," 2016). By making use of these API, developers can integrate mobile payment functionality in

their custom applications. This means that access to banking services can be extended to customers in a variety of ways. From making payments using debit and credit cards directly linked to their bank accounts to making direct deposits from their banks accounts to various merchant accounts.

In one developmental study (Ouko, 2013), different factors that play a vital role in the adoption of mobile banking by bank customers were identified. These factors are social, economic and technological. Social factors include but are not limited to the conceptualization of mobile money as a viable currency, awareness of the existence of these services, attitude towards changes in a new way of transacting, trust and level of confidence in one's bank or financial institution and the convenience of the service offered. Some economic factors include the cost of adopting this new means of service delivery and cost of purchasing a mobile device. Technological factors at play here include how secure this mobile banking service is, the service availability and reliability, privacy and ease of use just to name a few.

Adoption of mobile banking by banks and their customers is however not devoid of challenges. One of the most pressing challenges facing customers is the security of this relatively new service delivery channel (Ouko, 2013). How safe is the information sent from my device to the banks through unsecured networks and how secure is it against unauthorized access while in transit? This is the question that plagues many potential mobile banking customers. In turn, the burden of proving that this is a safe, secure and reliable mode of service delivery lies with financial institutions.

1.2 Attacks resulting from OpenSSL Vulnerabilities

Reported incidents of exploited SSL vulnerabilities such as the Heartbleed and Poodle bugs serve to further highlight the going concern of how secure mobile and internet banking services really are. The heartbleed bug is a critical flaw in the SSL code. It is located in the Heartbeat extension that allowed SSL based websites to establish and secure a connection with their users for an extended period of time (Codenomicon, 2014). This vulnerability enables an attacker to access the connection, read, capture and steal any and all data that is stored in the system's memory. Any such data can be emails, credit card information, passwords, instant messages.

For example, in April of 2014 when the Heartbleed bug in OpenSSL libraries was detected, the Canadian Revenue Agency reported that it had 900 social security numbers of Canadians stolen

from their online system (Acunetix, 2014). The stolen records could be used to gain access to government benefits or to perform identity theft. Even though they were able to take down their website soon after the bug discovery had been made public, the attackers still had a window of opportunity to steal sensitive data belonging to Canadian citizens.

American funds, a family of mutual funds from Capital Group that are sold primarily through financial advisors and intermediaries, were also affected by the Heartbleed bug (Mashable, 2014). They sent notifications to their customers to change their username and passwords. Venmo, a mobile payment service offered by PayPal, was also a victim of the heartbleed bug. They also sent out prompt notifications to their users informing them that the company had taken steps to patch the vulnerability before advising them to change their usernames and passwords.

1.3 Problem Statement

Active and potential mobile banking customers are concerned as to how safe and secure their personal banking information is when financial transactions are executed from a remote location and their financial details sent over a network from their mobile devices to the mobile banking servers.

Google (Google, 2016) discovered that developers are also making use of vulnerable third party OpenSSL libraries in their mobile applications to secure information being relayed from the customer's mobile devices to the servers. This makes their applications vulnerable to man-in-the-middle attacks.

Given the need to be ahead of the competition, organisations do not spend enough time during the development of the applications to focus on the security aspects required. Most of the time, application security is implied but rarely if ever is it tested before deploying the solution. This is because the main focus of such projects is meeting deadlines rather than providing secure systems (Rakitin, 1999). Mobile banking application development is not immune to such pressures. Applications are developed with the aim of getting an edge over competitors rather than providing secure applications.

1.4 Research Objectives

The following were the objectives of this study

1. To understand the mobile banking architecture and identify potential points of vulnerability.
2. To identify and understand the risks associated with using vulnerable third-party OpenSSL libraries.
3. To identify the available steps taken to address the use of vulnerable OpenSSL libraries.
4. To design, develop and test a tool that can be used to identify vulnerable OpenSSL libraries deployed in mobile applications and devices.
5. To validate proposed solutions by running tests on available mobile applications in the market place that have implemented OpenSSL libraries to secure communication.

1.5 Scope and Limitations

1.5.1 Scope

This study only sought to investigate how secure mobile applications were against OpenSSL vulnerabilities that contribute to fraud, hacking and phishing attacks. It looked at some of the most common vulnerabilities and how they can be detected during the product testing phase of the development cycle by means of the developed tool. Further, this study focussed only on the Android platform and Android driven mobile applications.

It did not look at how to repair the vulnerabilities detected, how to improve the manner in which projects are done in order to secure more time for product testing and the impact of internal fraud in a financial institution.

The testing framework was an Android application developed to run on devices running Android version 2.3 and above. Android studio provided the development and testing environment.

Even though this solution was tested on Kenyan mobile banking Android applications, it was also used to test how secure other kinds of Android driven mobile applications that seek to provide secure communication services for its users using the OpenSSL technology were.

1.5.2 Limitations

This focus of this study was primarily OpenSSL vulnerabilities. Any other identified vulnerabilities were noted and recommended for further study into vulnerabilities surrounding secure application development. It did not look into repairing or taking the necessary steps to

mitigate any of the identified vulnerabilities. The study only considered the Android environment and mobile applications deployed for Android devices.

1.6 Research Relevance

This study will provide financial institutions with a tool with which to test how secure their Android mobile banking application communications are. This should be able to give them the confidence to assure their customers that their mobile banking applications are secure. This tool can also be used to develop other Android-based applications that handle sensitive information and utilize OpenSSL libraries.

It will also seek to add knowledge on the vulnerabilities inherent in untested third-party libraries used in mobile applications development to provide crucial services such as network communication and authentication and how to test for OpenSSL vulnerabilities on these libraries. This information can be used by developers to develop applications that communicate and authenticate safely and securely. In addition, this study will also add to the foundation of knowledge being laid for research in mobile banking and its associated technologies as well as the need for efficient and effective project management with regards to development of software products.

Chapter 2: Literature Review

2.1 Mobile Banking Architecture

Mobile banking as defined by Ouko (Ouko, 2013) is the provision of banking and financial services through mobile devices such as smart phones and tablets over a telecommunication network. It makes use of Electronic Data Interchange (EDI) technology to transfer data in a form that allows automatic processing without requiring any manual intervention from bank employees. In a general mobile banking architecture, you have the main mobile banking application residing within the bank network or at the vendor's data centre (Paladion Networks, 2007). It communicates with the core-banking systems to forward transaction requests received from the customers. In order to communicate with the mobile banking application server, the mobile banking applications on the customers' devices send the transaction request through a public Internet Protocol (IP) that points to the application server environment. A dedicated leased link connects the mobile banking application with available mobile network operators through which confirmation SMSes on the transaction status are sent back to the customers.

Some mobile banking vendors also avail Unstructured Supplementary Service Data (USSD) alternatives to mobile applications. This is usually targeted to feature phones which are low-end devices not capable of running high-end mobile applications. The USSD alternatives are usually routed directly through the mobile network operators to the mobile banking servers through the dedicated leased link.

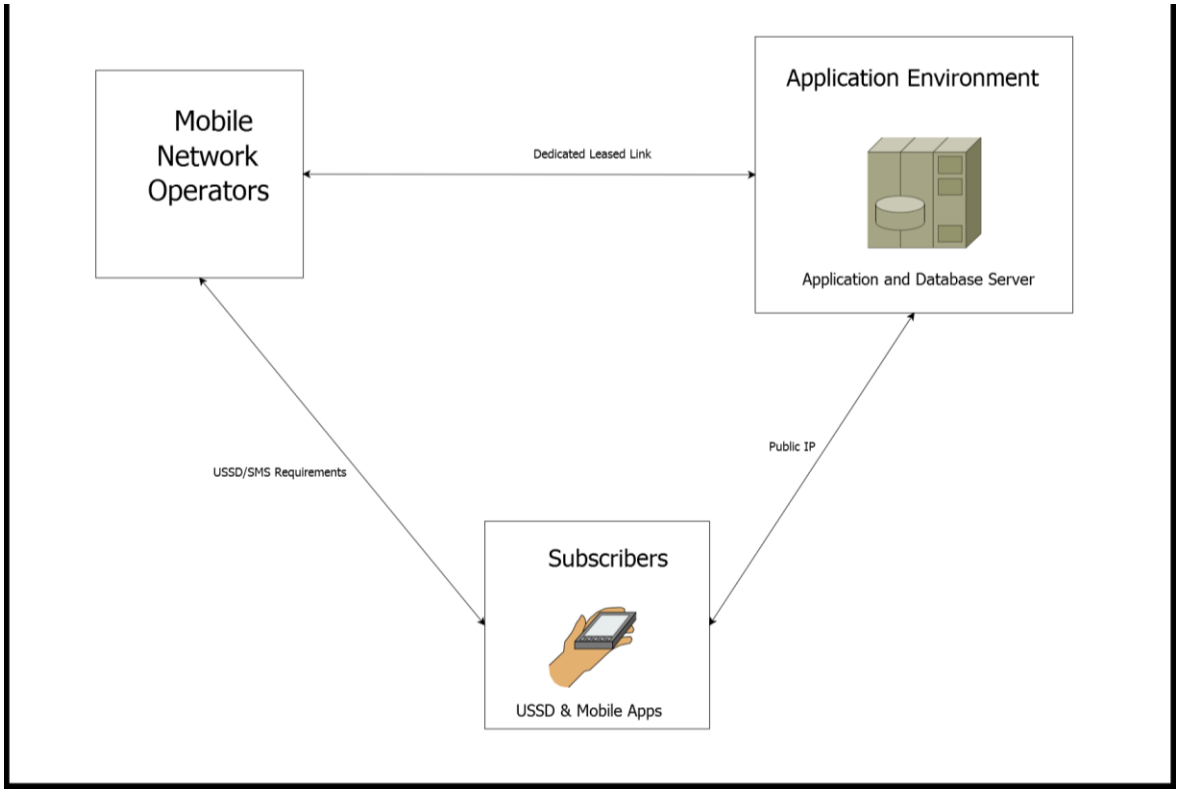


Figure 2.1 Mobile Banking Architecture inspired by Paladion (Paladion Networks, 2007)

In order to use mobile banking services, Paladion Networks (Paladion Networks, 2007) identified that bank customers must have their accounts activated for this service. It is at this point that their phone numbers are linked to their active bank accounts.

A good example of a financial institution that has made use of the aforementioned mobile banking architecture is Equity Bank, one of Kenya's largest financial institutions. As highlighted in their investors' presentation (Equity Bank, 2014), Equity Bank's mobile banking architecture has leveraged local mobile network operators like Airtel and Safaricom to provide their customers with access to the Equity Banking system through various channels one of them being their Eazzy Banking mobile application. As illustrated in Figure 2.2 below, once connected to their banking system, customers are able to enjoy the various financial, loans and savings products offered by the bank.

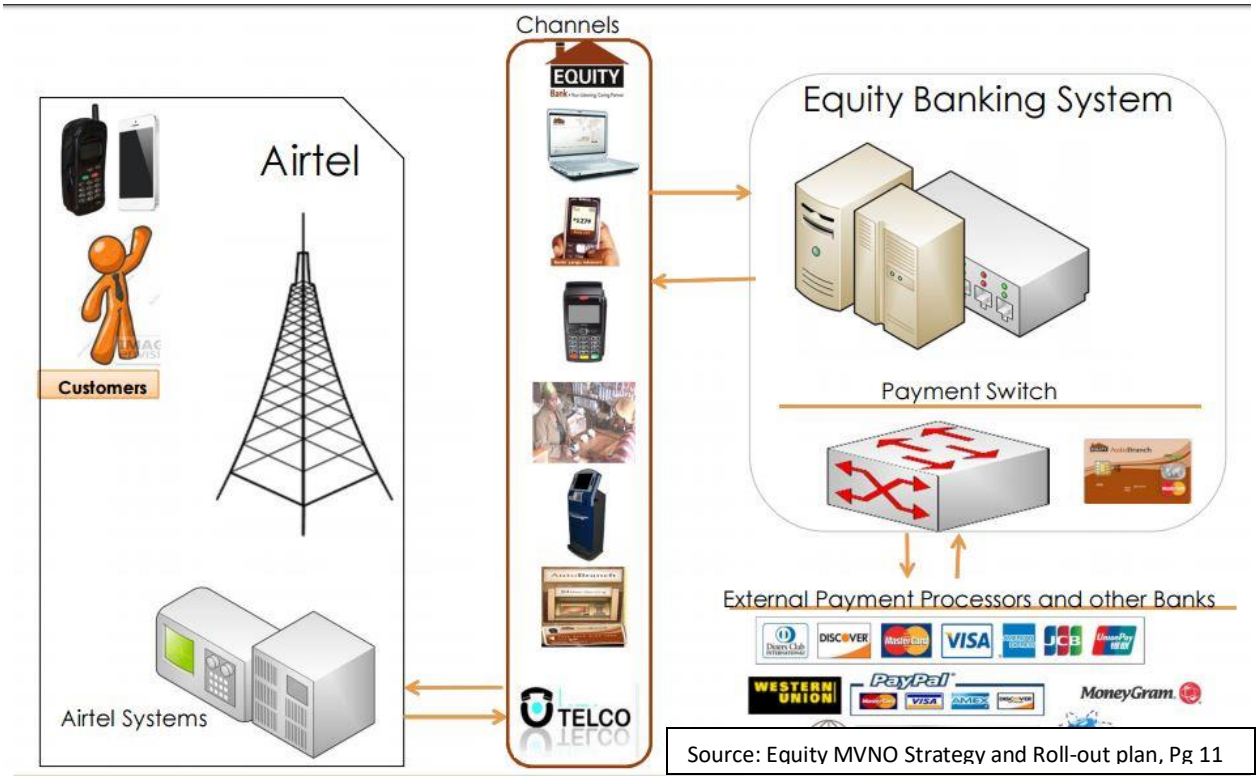


Figure 2.2 Equity Bank Banking services architecture

Some mobile banking providers install Secure Socket Layer (SSL) certificates on the public internet protocols to ensure secure data transmission over the medium. Secure Socket Layer certificates encrypt communication over the network using public/private keys between the client and the server.

2.2 Fraud cases in financial institutions

There has been an increasing rate of fraud, hacking and phishing attacks that are being reported world-wide. Kenyan financial institutions have also not been spared from this. According to Ouko (Ouko, 2013), potential mobile banking users may feel that performing banking transactions through their mobile devices rather than in person is very risky and not secure. Sending such sensitive information through unsafe wireless networks could compromise data privacy. Financial institutions are presently having challenges when it comes to cases of fraud with cheque and internal fraud cases being the major risks that banks in Kenya are presently grappling with (Guguyu, 2015). This does not bode well for the uptake of mobile banking services.

With mobile banking growing in popularity as well, mobile phones are being used as a means of authenticating bank customers without them having to physically be there at the financial institution. To do this, once they have downloaded the mobile banking application on their phone, mobile banking users are required to login using their registered credentials. Mobile banking applications are usually connected directly to mobile banking servers that can be reached through the internet. Sensitive identification information is then sent from mobile devices to the mobile banking servers over the internet. This expands the attack surface in banking systems by introducing potential areas of vulnerabilities that can easily be exploited by an unauthorized user and an internal member of staff (Temenos & NetGuardian, 2016). One such attack that can be deployed in such a setup is a man-in-the-middle attack. This is a cyberattack where someone with malicious intent can inject themselves into a communication channel between two unsuspecting parties. He then proceeds to impersonate both parties thus gaining access to information that the two parties are trying to send to one another (Gangan, 2015).

By performing a man-in-the-middle attack over an unsecured/compromised connection between a mobile banking application and the respective financial institution, an attacker can easily be able to retrieve the user's authentication information such as an identification number, phone number and passwords amongst other account details. He can then easily share this with an internal member of staff in the financial institution who has access to the client's data within the mobile banking system. The insider can then alter the credentials by putting the attacker's phone number as the active phone number of that account number. At this point, the attacker can call the bank's helpline requesting a password reset. This will be sent to the active phone number which now belongs to the attacker. He can then proceed to raid the account before having the insider revert the records back to the correct state. An attack influenced or perpetuated by an insider is very difficult to identify or recover from (Serianu, 2015). This is largely in part due to their knowledge of security and procedures already in place.

Using the acquired credentials and a little social engineering – an attack that relies on social interactions between people to trick them into breaking normal security procedures (Rouse, 2016) – the attacker can also launch a SIM swap attack on the unsuspecting victim. A SIM swap attack is an attack whereby the attacker uses your identification details and a little social engineering to convince your mobile network provider's agents at various stations and call

centres to activate a new SIM that they are in possession of. This deactivates the current SIM card that the unsuspecting victim has causing him to lose phone service (Brignall, 2016; Rajendran, 2014). The attacker can then proceed to make PIN change requests and perform account transactions through the mobile banking application on their phones using your mobile phone number and other identification details they already have. All responses from the bank are sent back to the attacker's phone since it currently holds the active SIM card registered to the victim. Given that the requests are coming from an authorized source (the victim's phone number), the financial institutions cannot flag this as an erroneous transaction attempt.

There is a thriving black market that deals with stolen customer information, including all online bank and credit card details ("Global Black Market for stolen personal data," 2016). The theft of confidential data from unsafe/compromised connections and other instances of fraud are damaging to a bank's image and reputation even if there was no direct loss of money (Temenos & NetGuardian, 2016). As it stands, fraud and identity theft is a serious concern for financial institutions and mobile banking related fraud can only make things worse.

2.3 OpenSSL Vulnerabilities

Chandra, Messier and Viega (Chandra, Messier, & Viega, 2002) describe OpenSSL as an open source cryptographic library that provides implementations on the industries best regarded encryption algorithms such as 3DES (Triple DES - Data Encryption Standard), AES (Advanced Encryption Standard) and RSA (named after its creators, Ron Rivest, Adi Shamir, Leonard Adleman). Due to the need to securely encrypt and decrypt data in motion, OpenSSL is designed to provide a secure means of communicating over an insecure medium. It is used in applications that need to secure communication going through networks against eavesdropping. It is also used in applications that need to verify the identity of the party at the other end. OpenSSL implements SSL and TLS (Transport Layer Security) protocols in an open-source environment.

In as much as OpenSSL strives to provide developers and administrators with a simple way of providing safe and secure network communication, in reality even the SSL protocol requires proper understanding of the underlying security principles in order to make use of it properly (Chandra et al., 2002). Due to this, numerous applications using OpenSSL are vulnerable to exploitation by attackers. This is largely in part due to poor understanding of security principles

by developers, vulnerable and poorly developed third party libraries used by developers and vulnerable security providers in-built in mobile devices.

There have been numerous instances of SSL vulnerabilities being exploited by attackers due to poor implementation of this cryptographic library. One such instance was during the incident of the Heartbleed bug that took the internet by surprise in April 2014. Found in the Heartbeat extension of OpenSSL that allowed websites to establish and secure a connection with their users for an extended period of time (Codenomicon, 2014), this vulnerability allowed attackers to read the memory of the systems that were protected by the vulnerable version of OpenSSL (Codenomicon, 2014). This makes it easy for attackers to eavesdrop on communications, steal data directly from services and users and also to impersonate them. The Canadian Revenue Agency was one of the first victims to come forward. 900 social security numbers were stolen as a result of the exploitation of this vulnerability (Acunetix, 2014). Other financial institutions that fell victim to this vulnerability were American Funds, a family of mutual funds from Capital Group and Venmo, a mobile payment service offered by PayPal (Mashable, 2014).

Several researchers (NIST, 2014; PCI Security Standards Council, 2015) have also published details on another security vulnerability that allowed attackers to perform a man-in-the-middle attack on a secure connection and extract data from it. It is most commonly known as a Padding Oracle on Downgraded Legacy Encryption (POODLE) bug and it was present in SSL version 3. To date, there are no known methods to mediate vulnerabilities such as the POODLE bug with early versions of SSL no longer meeting the industry standard security requirements. Suggestions of dealing with vulnerabilities on these vulnerable SSL versions are to update their applications to make use of Transport Layer Security (TLS) version 1.2.

2.3.1 Third Party Libraries

Mobile application developers tend to make use of third party libraries in their applications. These serve to provide additional functionality that the standard development environment cannot provide. They could be aesthetic modifications or even security based alterations. The latter is the more concerning. There are numerous mobile applications that make use of vulnerable third-party OpenSSL libraries to secure information being relayed from the clients to the servers. This makes their applications vulnerable to Man in the middle attacks. A recent audit carried out by Google (Google, 2016) on their mobile application store, Google Play store,

revealed that there were numerous Android mobile applications that had vulnerable deployments of OpenSSL. They were susceptible to the logjam and CVE-2015-3194 vulnerabilities. The logjam vulnerability works by enabling a man-in-the-middle attacker to downgrade the vulnerable TLS connections to using 512-bit export grade cryptography thus allowing the attacker to gain access to any data passing through the connection ("Weak Diffie-Hellman and the Logjam Attack," 2015). They could then edit the retrieved data as they saw fit. The CVE-2015-3194 vulnerability enabled attackers to decrypt Hypertext Transfer Protocol Secure (HTTPS) traffic.

Studies such as (Mettler, Zhang, & Raman, 2014; Peck, Kini, & Pyles, 2016) suggest that there was poor implementation of authentication certificate verification by developers in their Android applications. It was noted that mobile applications developers allow their applications to accept all incoming certificates, regardless of their certificate authority status. Out of 1000 of the most downloaded free apps on Google Play Store as of July 17th, 2014, it was found that of the 614 apps that implemented OpenSSL, 448 did not verify the authentication certificates they got from the servers. 50 of those applications did not authenticate the hostnames in the certificate. This means that the app can accept a certificate of one credible server issued to it by a different and malicious/compromised server that is launching a phishing attack – fraudulent requests that seem to come from legitimate sources with the intention of getting the target to reveal private information about themselves (Rouse, 2015) -at it.

In response to the audit that showcased the increasing cases of OpenSSL vulnerabilities in mobile apps in their Google Play app store, early this year (2016) Google begun sending alerts to the publishers whose applications contained vulnerable versions of OpenSSL (Google, 2016). The apps were still downloadable since Google could not block downloads without the publisher's authorization. However, updates still containing vulnerable versions of OpenSSL were not allowed. If publishers were using any OpenSSL deployments in their apps, the Google Play Store would only accept those apps with versions 1.0.2f/1.0.1r and above.

2.4 Time to market pressure

Time to market is the period of time from the conception of a product or a service to the time it is available for sale. The length of an organization's time to market for a product can directly affect the revenue the organization can generate from it ("Improving Time To Market," n.d.). The

longer it takes for you to create and launch a new product, the greater the opportunity your competitor has to grab a larger market share, leaving less overall revenue for you to go after when you eventually launch your product. Therefore, it is of great importance for an organization to optimize the time to market for its product offerings thus ensuring efficient resource management, predictable schedules and product launch date and increased total revenue on account of getting your product out to the market before your competition.

In 1999, Rakitin's (Rakitin, 1999) study on balancing the time to market and quality revealed that, for the sake of increasing the total revenue received from the sale of their products, organizations tended to rush their products to market with less regard for quality than for the timely release of their product. This is especially more pronounced with software products. The quality of software products is affected by, among other things, the customers' ever changing needs, the complexity of the system, the project teams' level of skill as well as management's investment in capacity building for its human resources. The manner of evaluating the performance of the development team also plays a part in the software development process and the quality of the final system. Most organizations have a performance evaluation system that is based solely on productivity rather that both productivity and quality. Developers are more concerned with meeting deadlines rather than providing quality systems. This is because their performance will be measured on how fast they are able to deploy systems.

With the focus being on meeting product release deadlines, developers and project managers are forced to schedule projects backwards in order to meet the time to market goals (Rakitin, 1999). Backward scheduling entails planning for a project based on the time available before the launch rather than the time required to develop the system. This leads to proper software development steps not being adhered to during development. Project complexities and unexpected occurrences are not taken into consideration, resulting in key deliveries either being late or not being delivered at all. This results in defective products that cause problems and dissatisfaction for their consumers. The organization then incurs expenses in repairing the defects. Resources that could otherwise have been engaged in the development of new cash generating projects are diverted to repair defects in deployed systems.

2.5 Discussion

From the available literature, it is clear that mobile applications in mobile banking systems communicate with the server through public IPs. This enables the bank customers to access banking services from anywhere in the world as long as they have internet access. In the event that this public IP is unsecured, attackers can easily eaves drop on the communication between the client and the server. Recorded incidents of OpenSSL vulnerabilities being exploited serve only to further question the security of mobile banking as a viable means of transacting.

There is also a gap in the understanding of security principles and their application by app developers. There are numerous mobile applications that have been deployed in the market place with vulnerable implementations of OpenSSL. This leaves the applications vulnerable to man-in-the-middle attacks that can result in attackers obtaining sensitive authentication and financial details belonging to customers.

Google has put forward a few recommendations on how developers can avoid developing vulnerable mobile applications. It has issued alerts through its mobile app store, Google Play Store, to publishers asking them to update their mobile applications if they had been flagged for having vulnerable OpenSSL implementations. This however doesn't address the applications that are already in the market place since they are still downloadable. Controls such as these are also not available in other application hosting environments such as the Amazon App Store, SlideMe, Yandex or vender maintained websites where they have personally hosted the applications available for download.

Chapter 3: Research Methodology

3.1 Methodology

This chapter documents the steps taken to address the objectives of this research. The selected approach is partly theoretical in nature through literature review and the development of a tool used to validate the proposed solution. A review of relevant research documents on mobile banking architecture, third party OpenSSL libraries was sufficient to fully answer the first three objectives of this study. The theoretical research was intended to provide a deeper theoretical understanding of the research area. This formed a basis for the development and validation of the proposed solution.

The tool developed is an Android mobile application that could be installed on any Android device running version 4.0 (Jelly Bean) or higher of the operating system. It scans the device for any installed application that makes use of third party OpenSSL libraries. An accompanying web portal provides version matching services by means of an API (Application Programming Interface). Various vulnerable OpenSSL versions were to be identified and registered on the portal with details of their vulnerability and a link will be provided where a user could get more information on it. By leveraging the services provided by the portal, the tool reviews the deployed OpenSSL version and compares it with registered versions flagged as vulnerable on the web portal. It then proceeds to indicate which vulnerabilities exist and the severity level each application identified contains before advising the mobile device user accordingly.

The severity rating given is based on the criteria used by CVE (Common Vulnerabilities and Exposure) ("OpenSSL: Vulnerability Statistics," 2017). It provides ratings for OpenSSL vulnerabilities by comparing the ease with which the vulnerability can be exploited and the impact it would have on the application given the domain of its deployment. The severity rating provided is either low, moderate or high based on comparison of the above.

The Software Development Life Cycle's (SDLC) Agile software development methodology was chosen for development. This methodology split the study into 5 distinct steps that are aimed at helping with creating software in short successive time frames called iterations. Agile is an adaptive approach that responds to changes very well, allowing for direct communication in

order to maintain transparency. It also helps to improve the quality of the software being developed by enabling one to find and fix defects in a timely fashion (Jamsheer, 2016).

In 2011, Waters' research on Agile Development Cycle (Waters, 2011) identified five key steps involved in this methodology. They are requirements analysis, design, development, testing and deployment.

3.2 Requirements Analysis

The initial phase of this methodology is the requirements analysis. It entails putting together all the requirements of the system to be developed ("SDLC - Waterfall Model," n.d.). The requirements for this research were obtained by through a critical analysis of the gaps that had been identified during the literature review. This formed the core of the requirements analysis. Additionally, the requirements were modified accordingly with each successive iteration due to the recommendations from previous iterations.

3.3 Design

After the requirements gathering and analysis phase comes the design phase that entails the interpretation of the requirements obtained from the previous phase.

Identifying the appropriate system design to be used in this research was a step-by-step process which was aimed at establishing the most fitting system architecture that would meet the identified requirements satisfactorily. From the interpretation of the requirements gathered in the previous phase, it was evident that the best system design to implement was a client-server architecture. This enabled the researcher to develop a system that ensured the registered comparison data was updated and managed from a central source to be consumed through the internet by the matching tool that would be deployed on mobile devices.

Users of the tool would require internet connectivity to be able to perform successful vulnerability matching.

3.4 Development

Development being the next phase, is concerned with coming up with small functional units with each successive iteration. For this research, this phase consisted of developing the various modules of the system. The tools used included java for the development of the Android

application tool. For the web portal, PHP was used as the development tool with MySQL providing the database environment.

3.5 Testing and Deployment

The next phase, testing and deployment, oversees the integration of all system components and their subsequent testing. To this end, a testing team of 20 users was assembled to test the Android application tool while a team of 3 users was set up to test the functionality of the developed web platform. The Android app testing team was composed of Android device users of varying ages, social standing and preference in mobile application choices. This was done with the aim of ensuring that the results of the scan covered as many categories of mobile applications as possible.

The mobile application tests were split into three phases that evaluated different aspects of the tool. The phases were installation testing, basic functionality testing and usability testing. Installation testing was concerned with verifying the process of installing, updating and uninstalling of the mobile application tool on the users' devices. Basic functionality testing looked at whether the app met the minimum requirements for the required functionality. Usability testing looked at whether the application performed properly and achieved its set goals.

For the web portal, the 3 users selected were familiar with web technology and updating content through the internet. The test for the web portal were split into various phases covering functionality, usability and compatibility tests.

The testing period for both teams was set to last 2 weeks with feedback from each team being received and incorporated into each iterative deployment of the system.

Chapter 4: System Design and Architecture

The solution was developed using a client-server architecture. An online portal/web application was developed as a client that provides a means of cataloguing identified vulnerable versions of OpenSSL. A mobile application was also developed as a client and its role was to scan a mobile device, identifying installed applications and then scanning these applications for OpenSSL libraries. If an OpenSSL library is detected in the selected application, the scanning application would send a request to the server to check if the version of the detected OpenSSL version has been flagged as vulnerable.

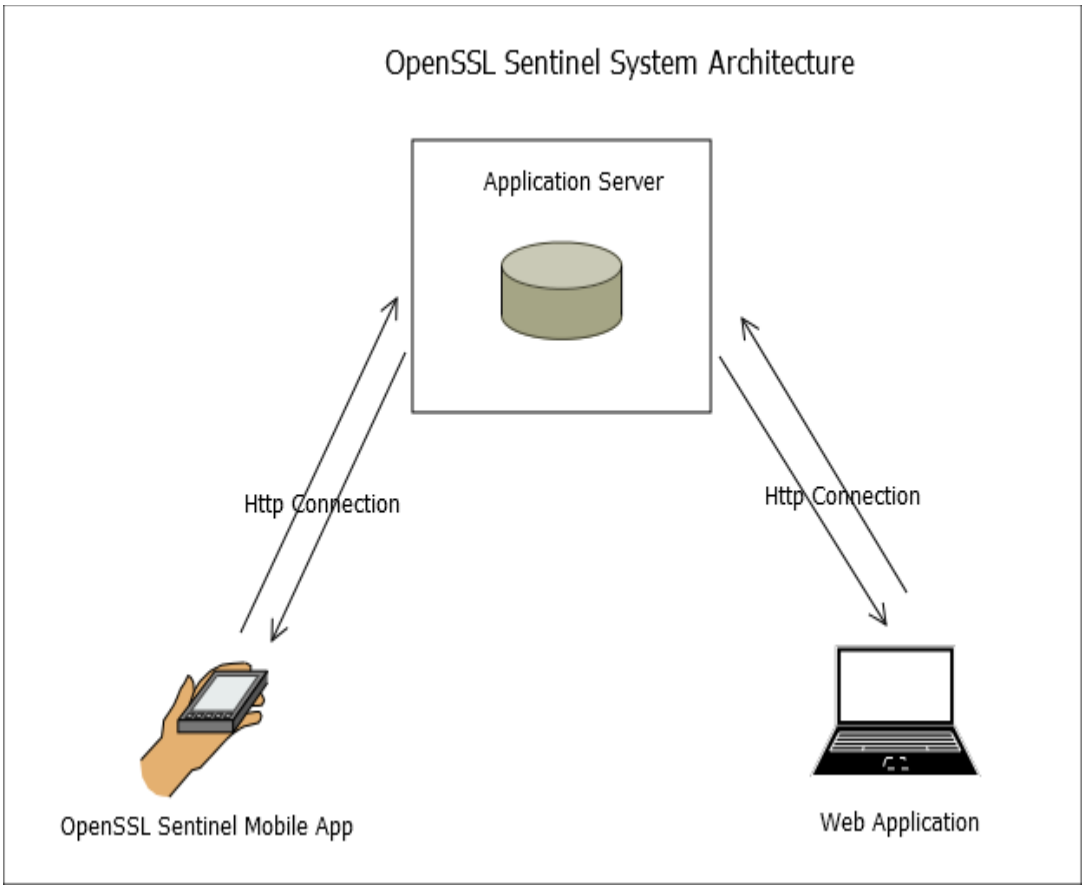


Figure 4.1 Client server system architecture

4.1 The Web Application

A web application was developed to provide a means of cataloguing vulnerable OpenSSL versions and providing further details of the vulnerability as well as recommendations on how to deal with it and a link to where more information on it can be found. Administrative users who update these records could also be created through the web application.

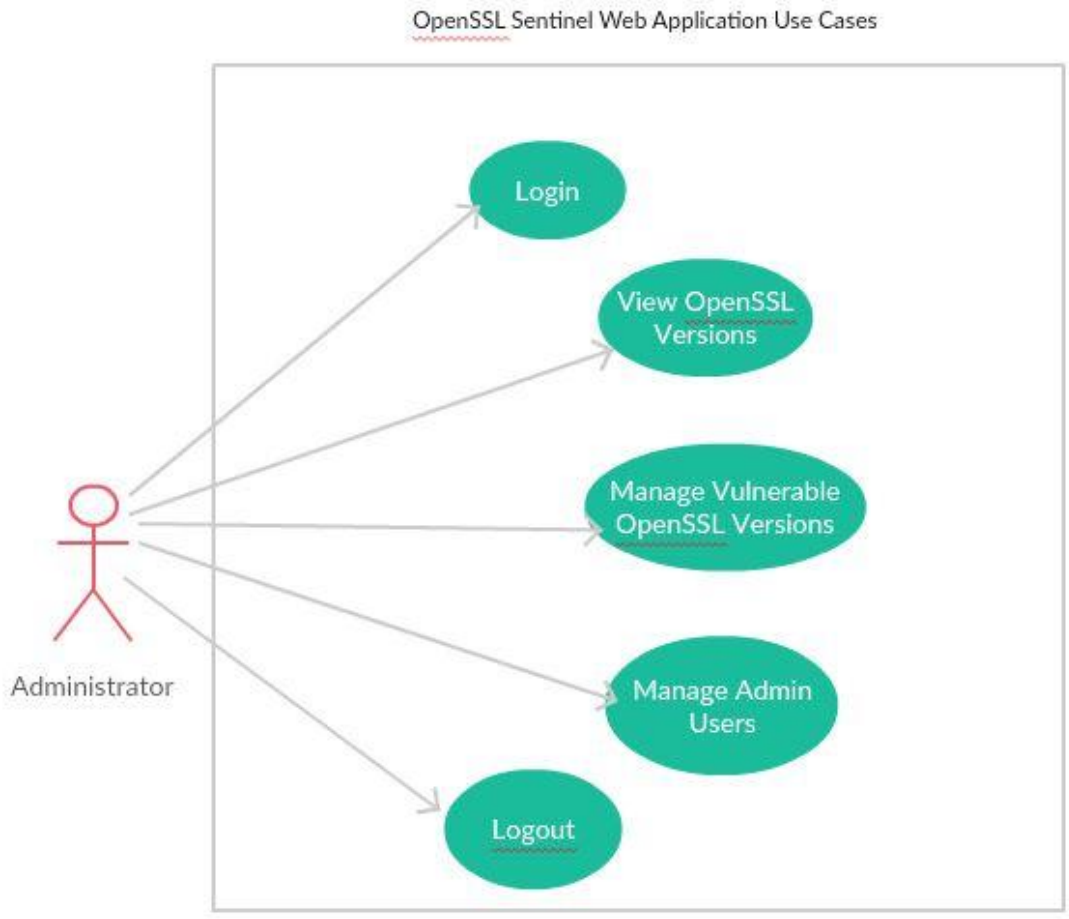


Figure 4.2 Web Application Use Case

4.1.1 Database Schema

The database schema that was applied in designing the server contained three key tables to hold the details of the vulnerable OpenSSL versions. The OpenSSL version table holds details of the various OpenSSL versions, the vulnerabilities table holds information about registered vulnerabilities and the version vulnerabilities table holds the data of identified vulnerabilities and the OpenSSL versions that are susceptible to it. It also holds on how to handle applications that had vulnerable deployments of OpenSSL and a URL (Uniform Resource Locator) link that directed the user where they could get more information on this vulnerability.

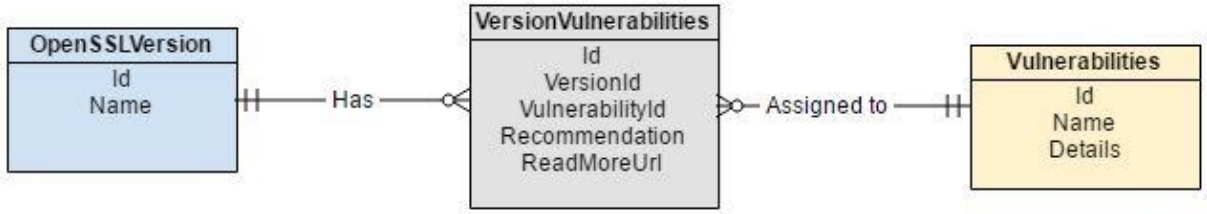


Figure 4.3 Entity Relationship Diagram

Below are the database table descriptions of the server application

Table 4.1 System Users

Field	**Data Type**	**Details**	**Notes**
id	Int	AI, PK	Auto Increments with every new user registration. It also serves as the primary key for each record
username	Text		
Emailaddress	Text	UNIQUE	Each active user should have a unique email address that is used during login.
password	Text		

Table 4.2 Vulnerabilities

Field	Data Type	Details	Notes
Id	int	AI, PK	Auto Increments with every new record added. It also serves as the primary key for each record
name	Text	UNIQUE	The registered name of the vulnerability
details	Text		Details on the vulnerabilities

Table 4.3 OpenSSL Versions

Field	Data Type	Details	Notes
Id	int	AI, PK	Auto Increments with every new record added. It also serves as the primary key for each record
name	Text	UNIQUE	The registered name of the openssl version

Table 4.4 Version Vulnerabilities

Field	Data Type	Details	Notes
Id	int	AI, PK	Auto Increments with every new record added. It also serves as the primary key for each record
VersionId	int	FK	Foreign key to OpenSSL version table
VulnerabilityId	int	FK	Foreign key to vulnerabilities table
Recommendation	Text		Recommendations on how to deal with this vulnerability
ReadMoreUrl	Text		Link to more details

4.1.2 Interface Design

Since this was a prototype system, the server application only has 3 main interfaces that the user can interact with. These interfaces are login, system user listing and vulnerabilities listing. The interface design of the server application was designed to be intuitive and easy to use for the user. The wireframes below provide the skeletal frame work of the web application.

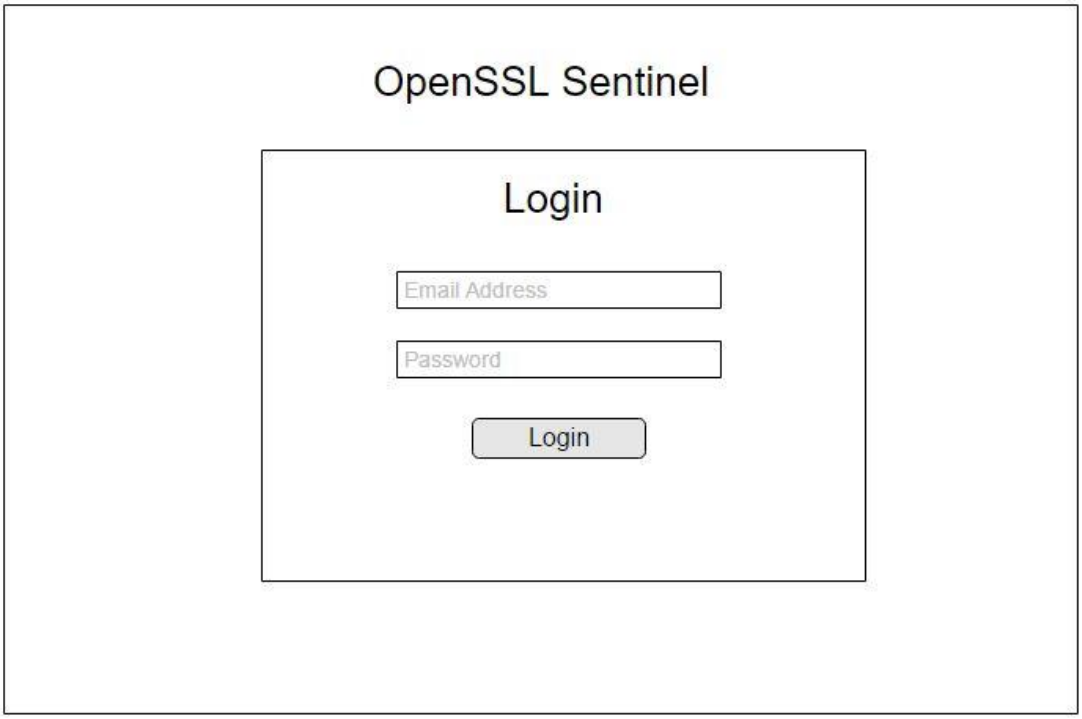


Figure 4.4 Login Screen

Figure 4.5 Vulnerability Registration

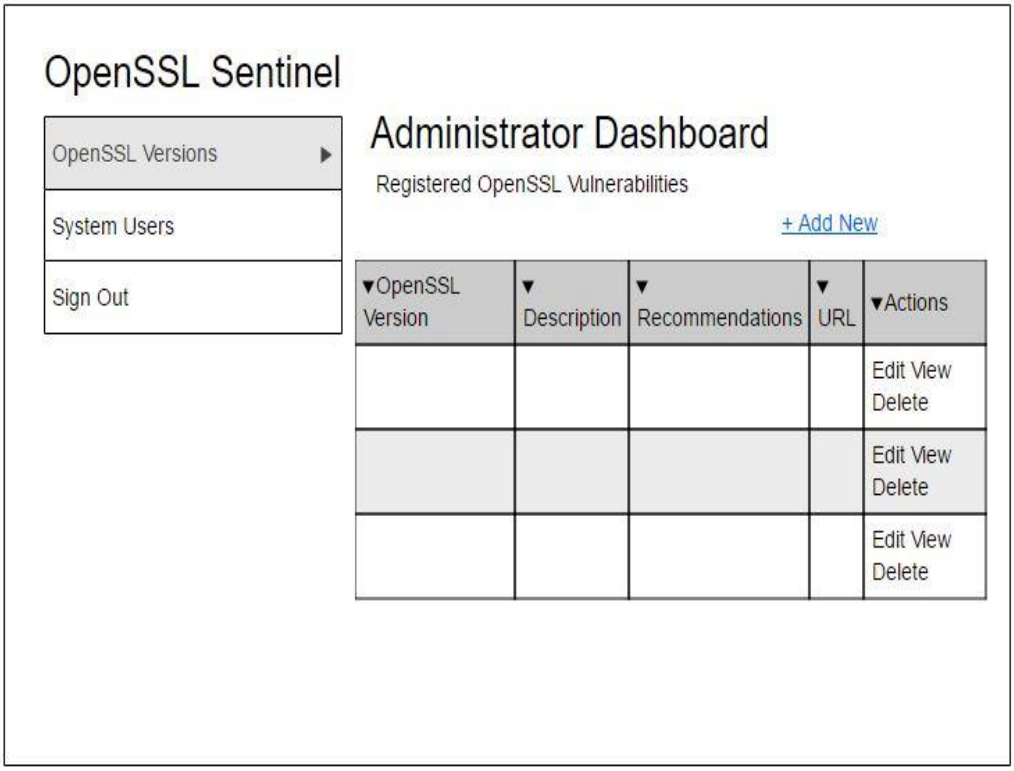


Figure 4.6 Vulnerability Listing

Figure 4.7 User Creation

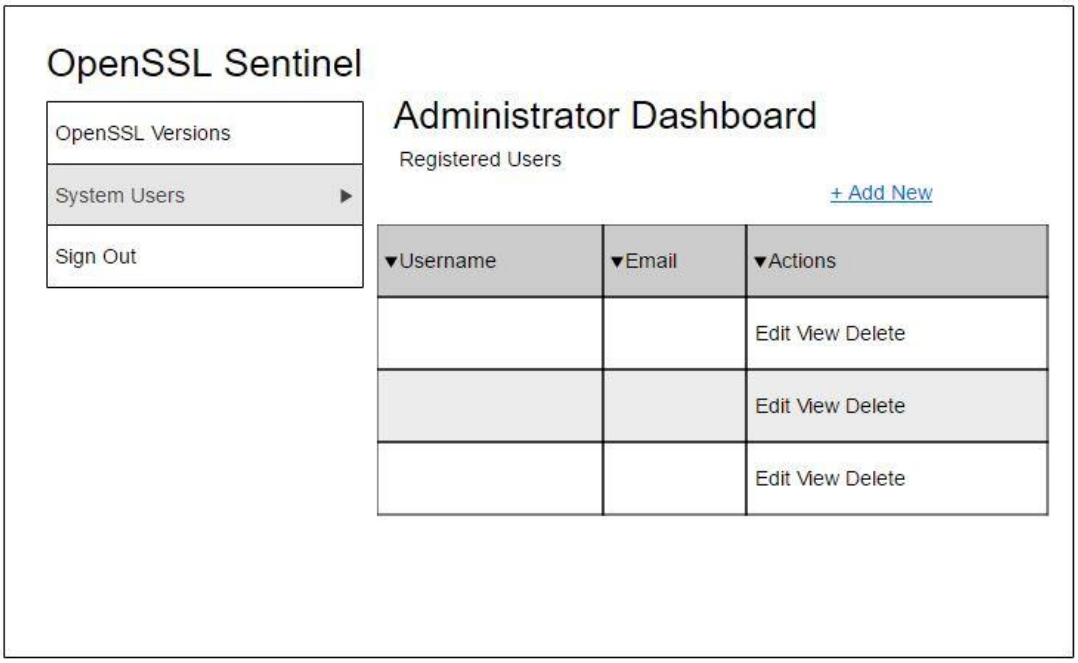


Figure 4.8 System User Listing

4.2 The Mobile Application

The mobile application was designed to run on Android devices with an operating system version 4.0 and above. There was no database deployment running on the mobile application because all records on registered vulnerabilities were maintained on the server. This ensured that the application was fetched the latest reviews of catalogued vulnerabilities. From the mobile application, a user could be able to view the scan results of the device's OpenSSL version and a listing of all installed applications on the device. They could also proceed to select an application to perform a scan on as well as see detailed results of the scan.

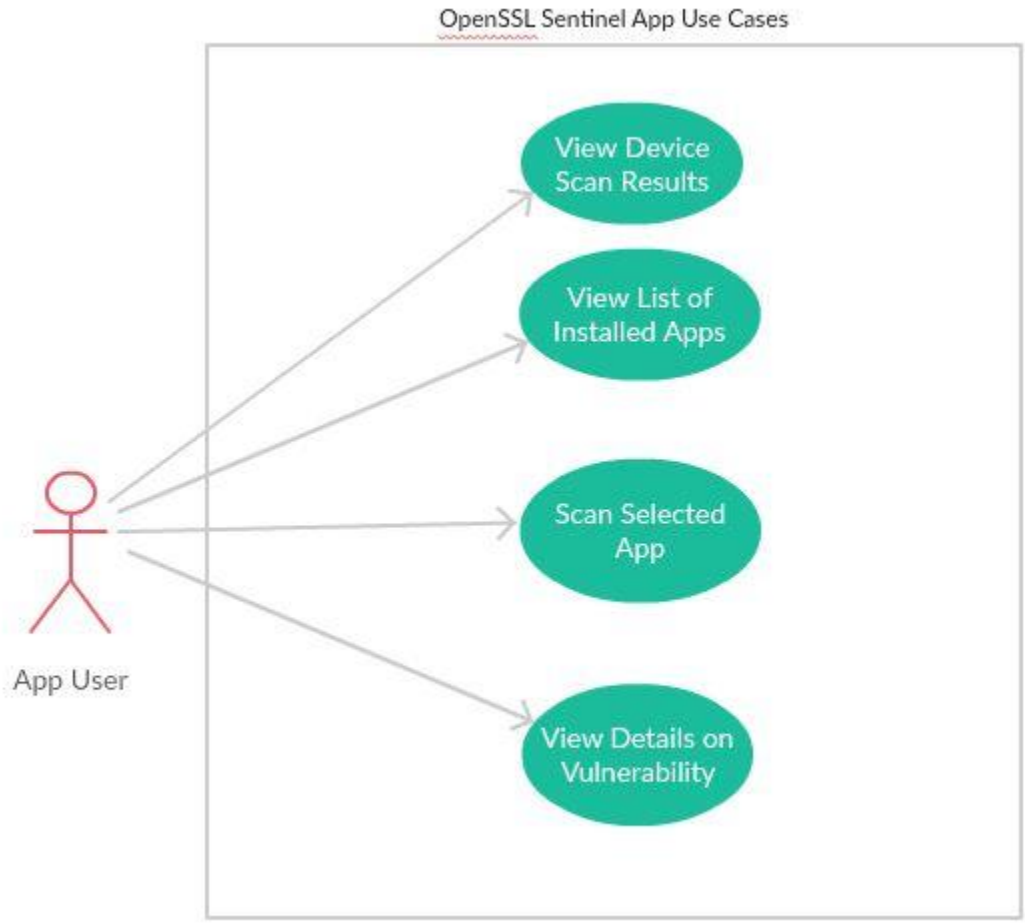


Figure 4.9 Mobile Application Use Case

4.2.1 Interface Design

The design was also intuitive in nature, making use of design principles recommended by Android. On launching the application, a listing of all applications installed on the device was provided. The results of a device scan to locate any inherent SSL versions are also displayed.

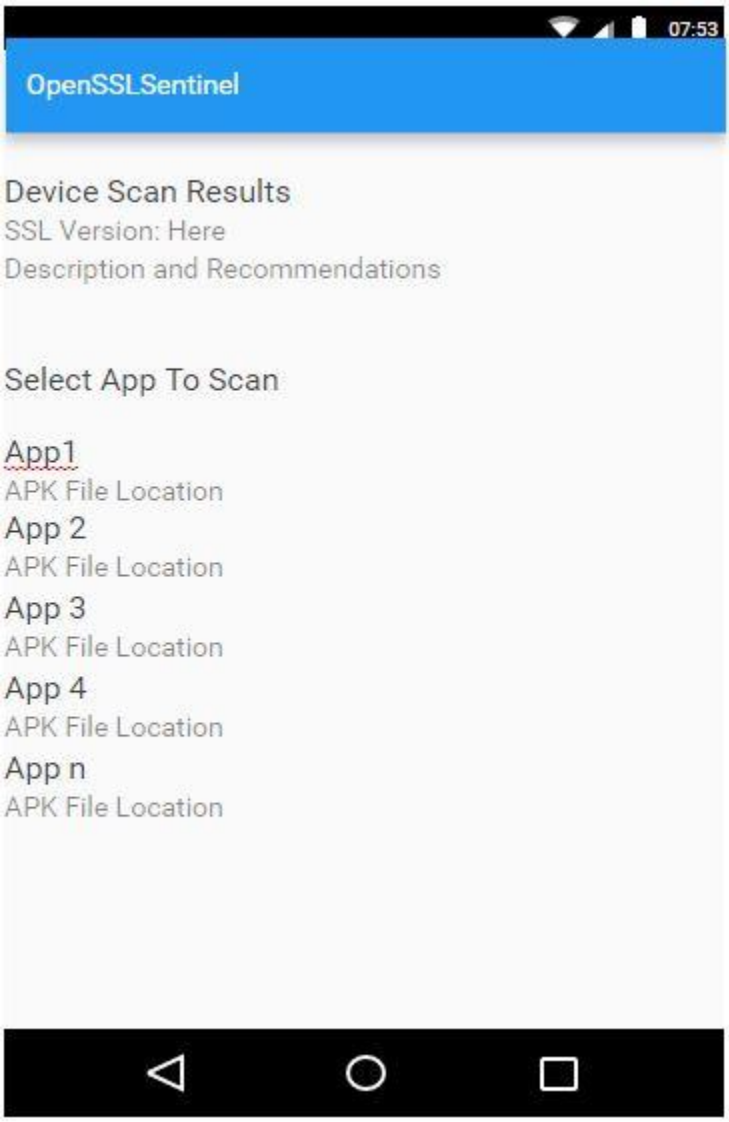


Figure 4.10 List of mobile applications installed on the device

On selecting an application, a scan is performed to check whether it has any OpenSSL library deployments and the results are displayed for the user's consumption.

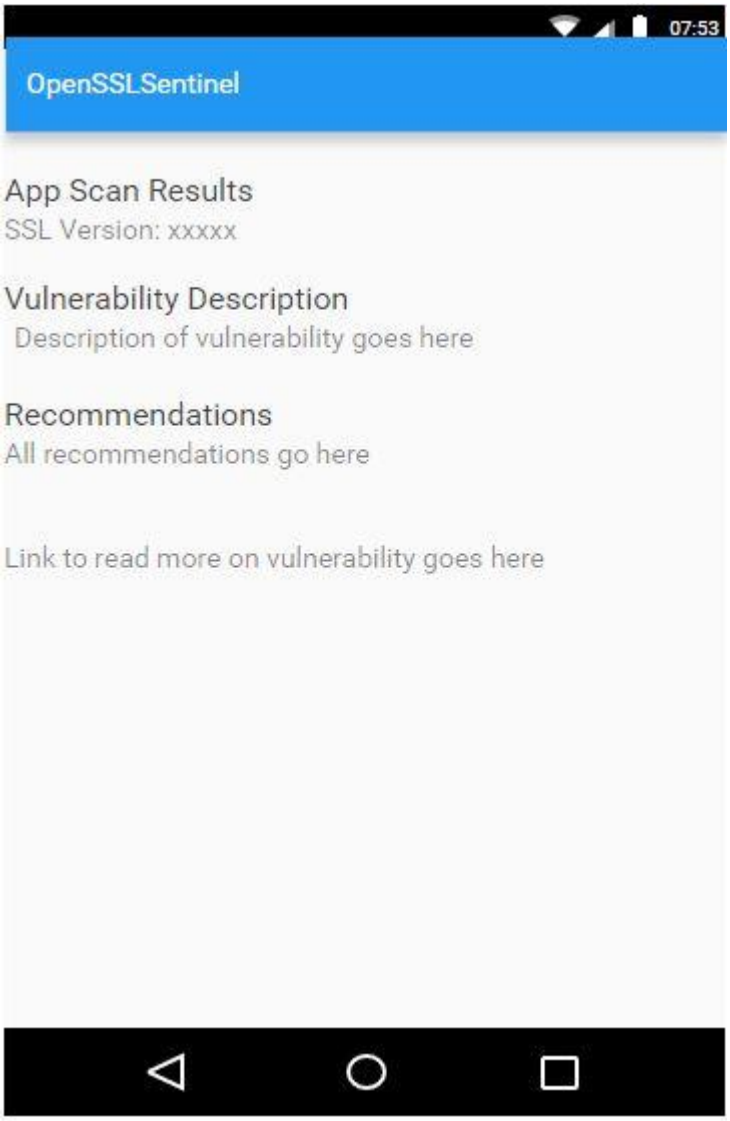


Figure 4.11 Results of a scan on a selected application

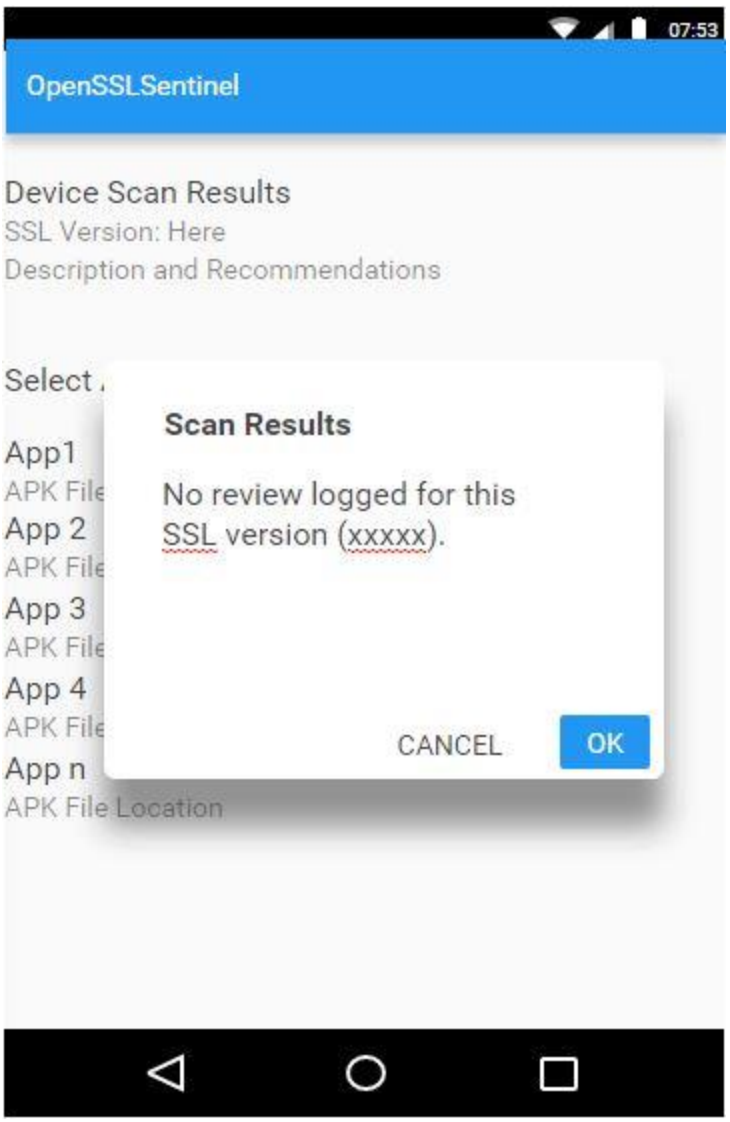


Figure 4.12 Results of a failed scan on a selected application

Chapter 5: System Implementation and Testing

5.1 System Implementation

5.1.1 Vulnerability Registration

Once logged in to the online application, the administrative user was able to register vulnerable OpenSSL versions. To do this, they were required to provide the version of the OpenSSL library in question, a description of the vulnerability/vulnerabilities it is susceptible to, recommended steps to take when dealing with this vulnerability and a link from where a user can get more information on the vulnerability.

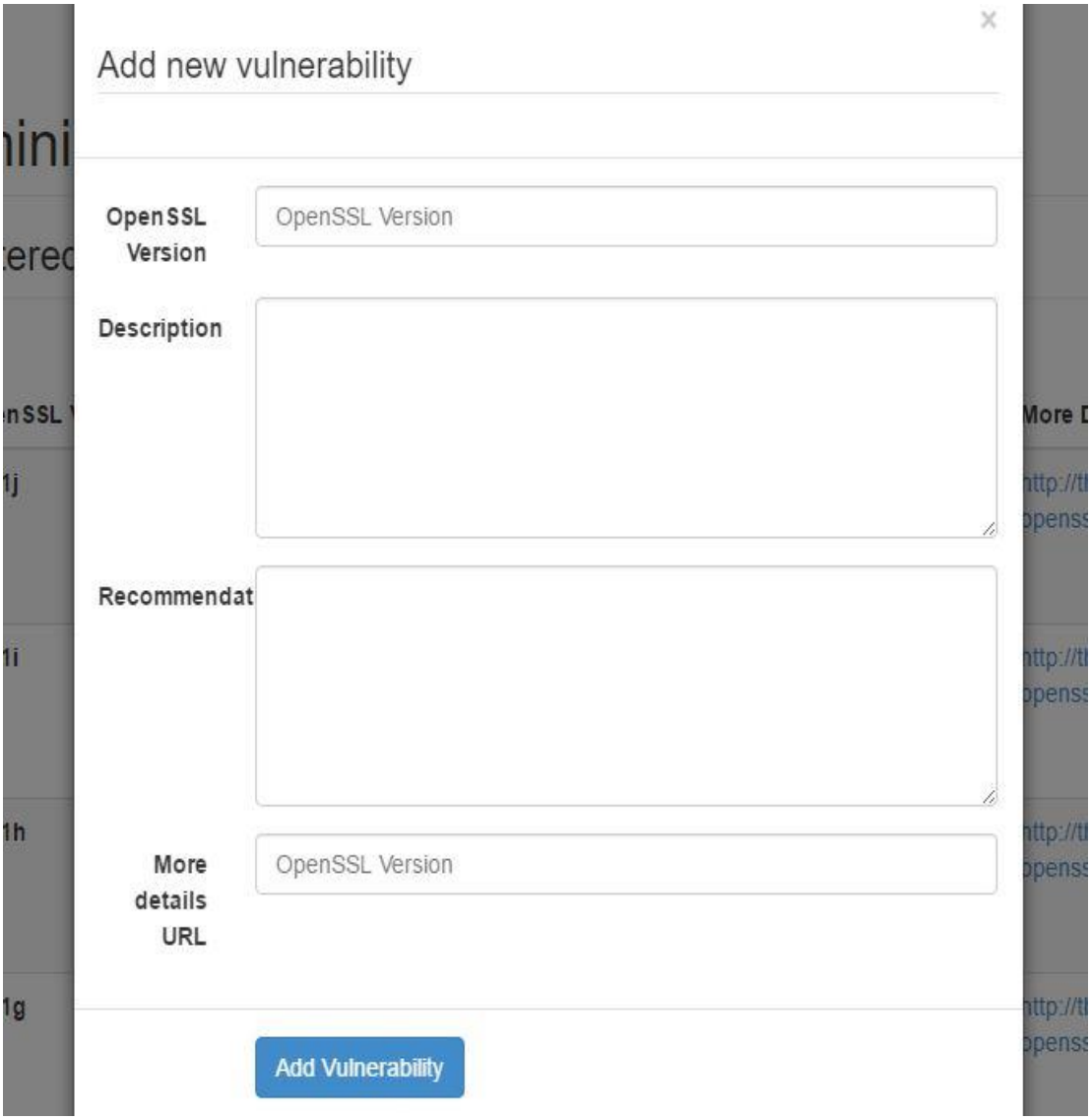
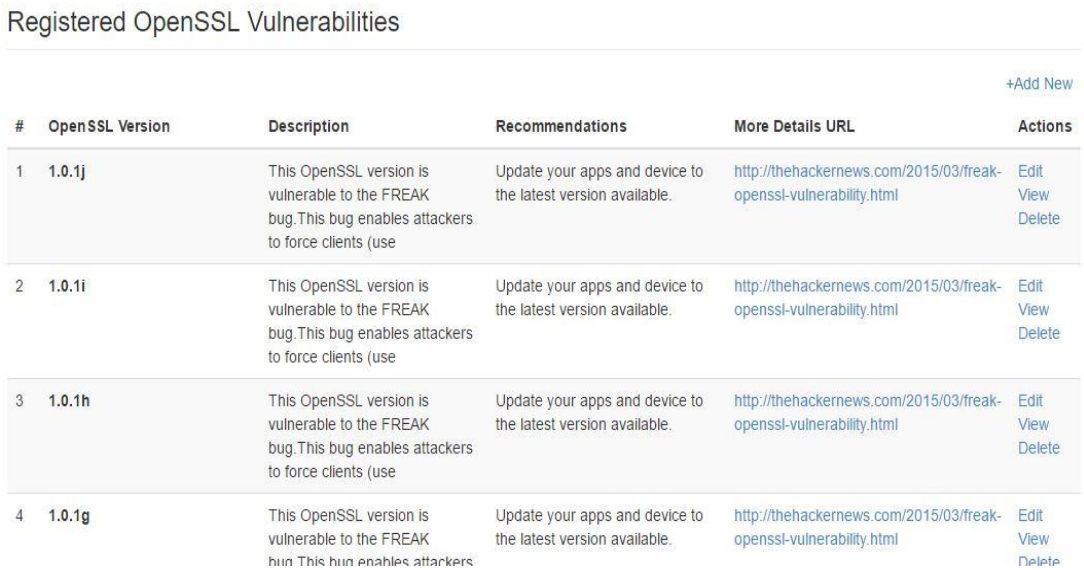


Figure 5.1 Registering a vulnerable OpenSSL version

All registered vulnerable OpenSSL versions were listed on the web application and could be updated or deleted at will. Their details were also visible by selecting the view action from the options on every registered OpenSSL version.



Registered OpenSSL Vulnerabilities

+Add New

#	OpenSSL Version	Description	Recommendations	More Details URL	Actions
1	1.0.1j	This OpenSSL version is vulnerable to the FREAK bug.This bug enables attackers to force clients (use	Update your apps and device to the latest version available.	http://thehackernews.com/2015/03/freak-openssl-vulnerability.html	Edit View Delete
2	1.0.1i	This OpenSSL version is vulnerable to the FREAK bug.This bug enables attackers to force clients (use	Update your apps and device to the latest version available.	http://thehackernews.com/2015/03/freak-openssl-vulnerability.html	Edit View Delete
3	1.0.1h	This OpenSSL version is vulnerable to the FREAK bug.This bug enables attackers to force clients (use	Update your apps and device to the latest version available.	http://thehackernews.com/2015/03/freak-openssl-vulnerability.html	Edit View Delete
4	1.0.1g	This OpenSSL version is vulnerable to the FREAK bug.This bug enables attackers	Update your apps and device to the latest version available.	http://thehackernews.com/2015/03/freak-openssl-vulnerability.html	Edit View Delete

Figure 5.2 Listing of vulnerable OpenSSL versions

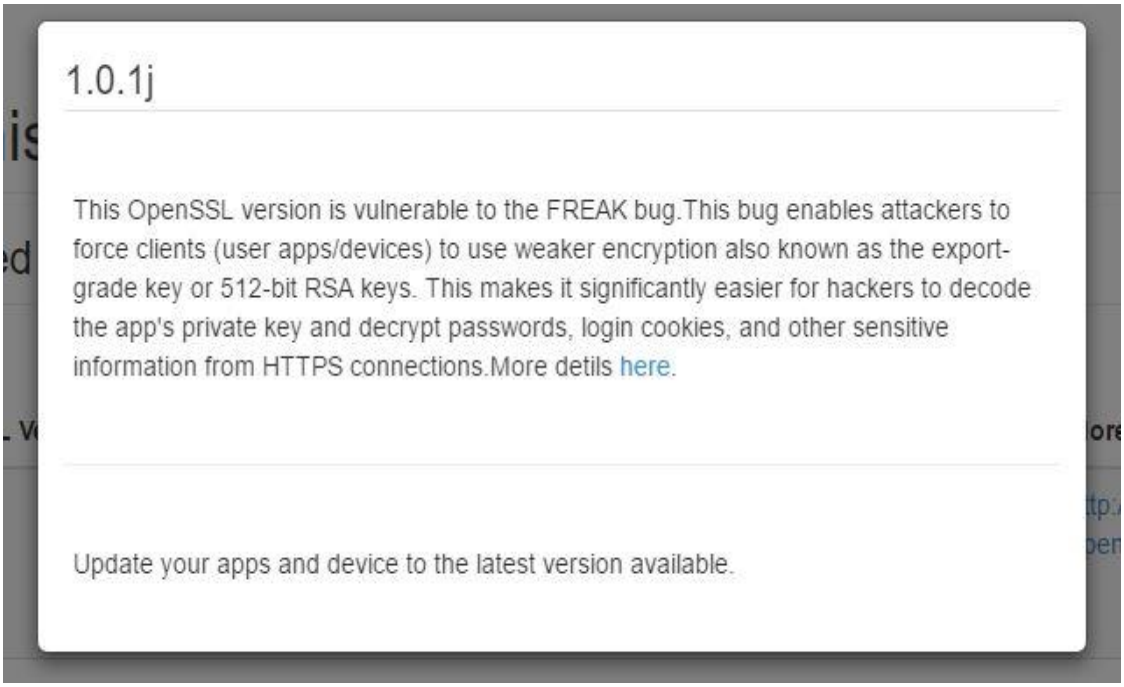


Figure 5.3 Sample details of a vulnerable OpenSSL version

5.1.2 Device and Application Scanning

Once the mobile application had been installed on an appropriate device and launched, it performed two initial scans on the device. The first scan was to see if the device was bundled with an OpenSSL library as is the common practice with all Android devices. This library is usually stored in the Android device directory found at "/system/lib/libssl.so". Once an OpenSSL library was discovered, a request with that OpenSSL version was sent to the server to check if it had been flagged as vulnerable. The results were then returned and displayed to the user.

The second scan was launched immediately after the initial device scan. This second scan checked for all third-party applications installed on the device. These were all the applications manually installed by the user from either the google play store or other mobile application repositories. They were then populated in a list view and displayed to the user with a prompting to select which application they wanted to scan for vulnerabilities.

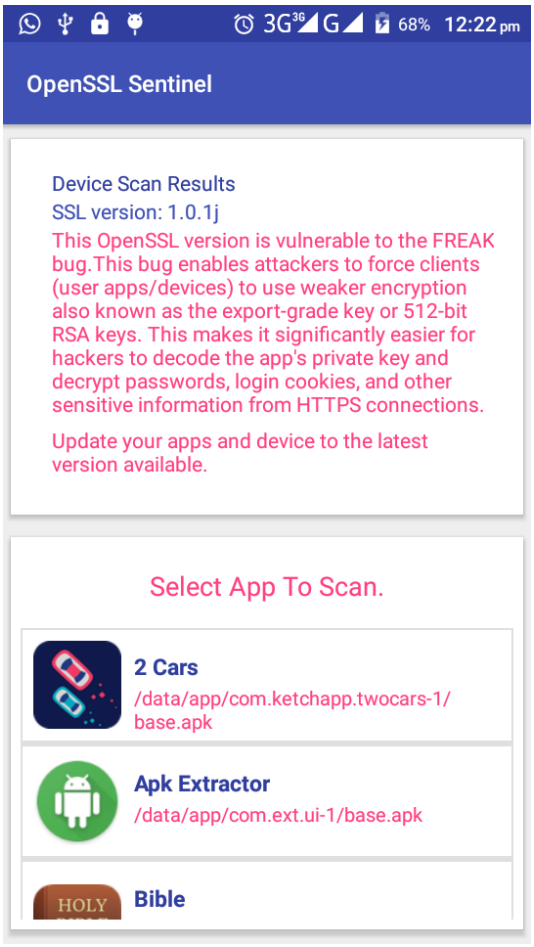


Figure 5.4 Device scan results and third-party app listing

5.1.3 Vulnerability Matching

To perform a successful vulnerability match on the selected application, the tool had to first of all locate the ".apk" file of that application as it was stored in Android device's public source directory. Once it had located the file path to the ".apk" file, the tool proceeded to recreate the file in its own local directory. This enabled it to have full read, write and execute rights on the created file.

With the new package file created in its resident directory, the tool loops through each file entry contained in the package file looking for a file with an extension of ".so". These are binary file formats in which libraries are saved in the Android applications. Once located, the tool would then proceed to read the data section of the binary files into a byte array. It does this because OpenSSL libraries embed their version in the data section as a string. A UTF-8 encoded string representation of the data is then created using the byte array.

In the library, the OpenSSL version data is prefixed by the phrase "part of OpenSSL" and suffixed by a date indicating when it was deployed. The date takes the format of "dd MMM yyyy". The tool makes use of the parsing function "indexOf" to locate the existence of the prefix phrase in the string representation of the data. This parsing function returns a whole number that indicates whether or not it has located the prefix phrase. If the index returned is greater than zero, then the prefix phrase has been located and the tool proceeds to extract the text between the prefix phrase and the suffix date. It does this by employing a string function that extracts the characters from a string between two specified indices returning a new string. In this case the indices are the prefix phrase and the suffix date. The new string returned represents the OpenSSL version number.

This number would be sent to the server where it would be compared with the registered vulnerable OpenSSL versions. In the event of a positive match, a notice would be sent back to the user with details on the identified vulnerabilities.

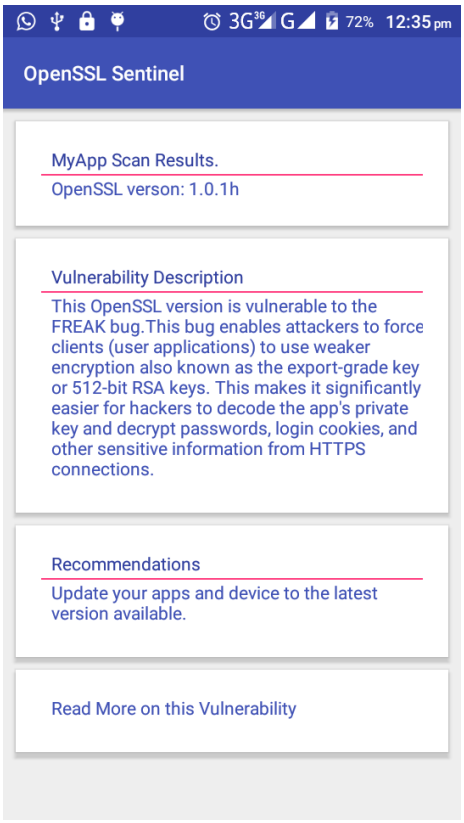


Figure 5.5 App scan results

5.2 System Testing

System testing was conducted in sessions where the selected participants were given a chance to fully test the assigned components of the system, i.e. the web component of the system and the mobile component of the system.

5.2.1 Web application testing

For the web testers, they were given the following link to access the system that had been hosted in a live environment. http://opensslsentinel.niqotechsystems.com.

5.2.1.1 Functionality Tests

Functionality tests were meant to check if the system had met its functional requirements. The web testers identified that all the links that had been provided for navigation within the system were working properly and were navigating the user to the correct section of the web platform. There were no broken links that were highlighted.

Submission forms were also noted to be working as expected. When registering new vulnerable versions of OpenSSL or while creating new users, the appropriate error messages were provided in the event that a mandatory field was left empty when submitting data. Once submitted, new records appeared in the listing with all other recorded information signifying successful record creation. One could also be able to edit or delete records by accessing the appropriate links to this functionality.

5.2.1.2 Usability Tests

With regards to usability testing, the testers found that the navigation menu and other system links that navigated to different pages and functions were easily visible and consistent on all web pages. Given that the content was user generated, the system maintained the data as it had been uploaded. No changes were made to the data by the system unless initiated by the user. This ensured data integrity when the data was saved in the system.

The testers also noted that the design was intuitive, ensuring an easy learning curve for the user. There was a constant theme throughout the website. Create, edit, view and delete links were all found in the same place and the navigation menu was also in the same place in all web pages. The testers were especially pleased with the additional benefit of having the menu item highlight the link to the currently active web page. This enabled them to know on which page they were currently at.

5.2.1.3 Compatibility Testing

The testers were able to ascertain that the website could display properly on a number of browsers, namely Chrome, Firefox, Opera, Internet explorer and Microsoft Edge. It was however noted that there was a slight aesthetic difference in rendering of web elements such as buttons, links and text field among the various browsers when viewed in different operating systems. This difference was however merely aesthetic and did not affect the performance of the system.

It was noted by the web testers that the website was not compatible with mobile and tablet browsers. It did not display correctly and the content appeared squeezed and unclear.

5.2.2 Mobile Application Testing

For the mobile application testers, the following link was provided from where they could download the mobile application to their devices. http://opensslsentinel.niqotechsystems.com/OpenSSLSentinel.apk. To this end however, they

were required to access their device's settings and allow the installation of apps from unknown sources. This was because the application was hosted in a separate environment and not the Google Play Store.

5.2.2.1 Installation Testing

Among the testers of the solution who were chosen to test the mobile application, even though they opted to carry on with the texting exercise, 100% of them showed great concern at having to allow their devices to install applications from unknown sources. They feared this would make their mobile devices vulnerable to other security risks.

Once installed, the testers reported that the application went through same installation process as other applications installed for the Google Play Store. This process involved disclosing the permissions the application was requesting and allowing the user to decide whether or not to install the application. Once the application was installed, the testers reported that they were prompted whether they should launch the application immediately or not. This is the typical behaviour displayed when installing application from the Google Play Store.

Once the testing period was over, 35% of the testers chose to uninstall the mobile application tool from their mobile devices. They reported that the uninstallation process went well and without any issue. It uninstalled much like other applications downloaded from the Google Play Store.

5.2.2.2 Basic Functionality Testing

100% of the testers of the mobile application reported that they were able to get a successful scan result of their device on launching the application. The results of the initial scan were able to indicate the OpenSSL version that had been deployed on their devices and a listing of all the applications the testers had installed on their devices. 75% of the testers had internet connectivity on their devices when running the initial tests either through a wi-fi access point or cellular internet. They reported that the scan of their device's OpenSSL library showed more details on the deployed version. The details also included recommendations on corrective actions to take.

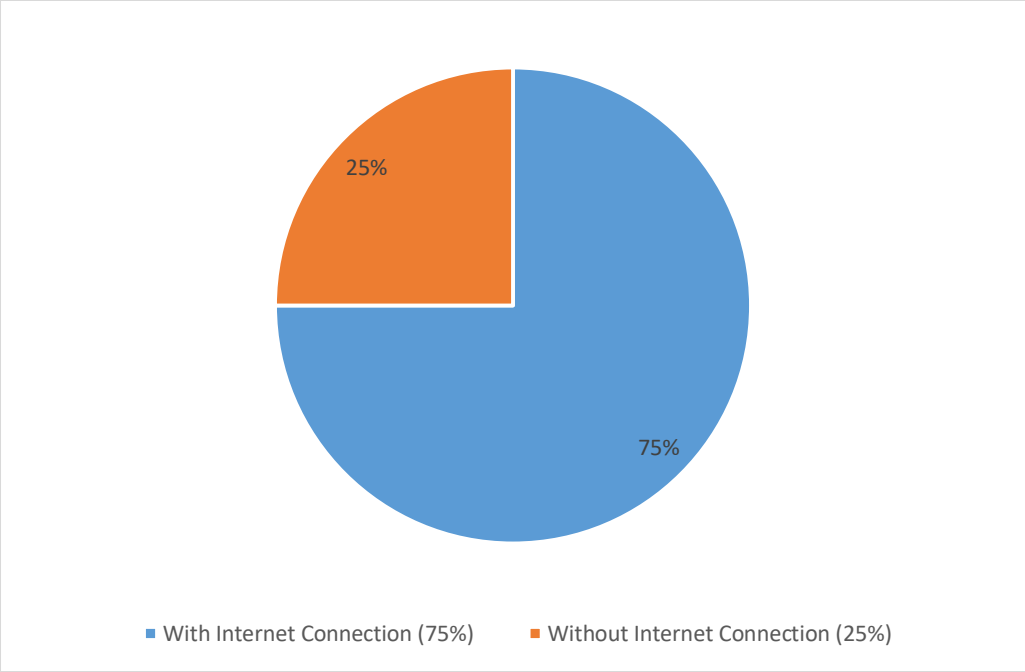


Figure 5.6 Comparing internet connectivity among the testers

The testers were also able to launch scans on the installed applications for vulnerable OpenSSL versions. 70% of the testers reported receiving feedback on vulnerable OpenSSL versions deployed on selected applications. It was noted that most of the applications that were found to have vulnerable deployments of OpenSSL libraries belonged to the gaming category. Other identified categories were entertainment, social media and chatting applications.

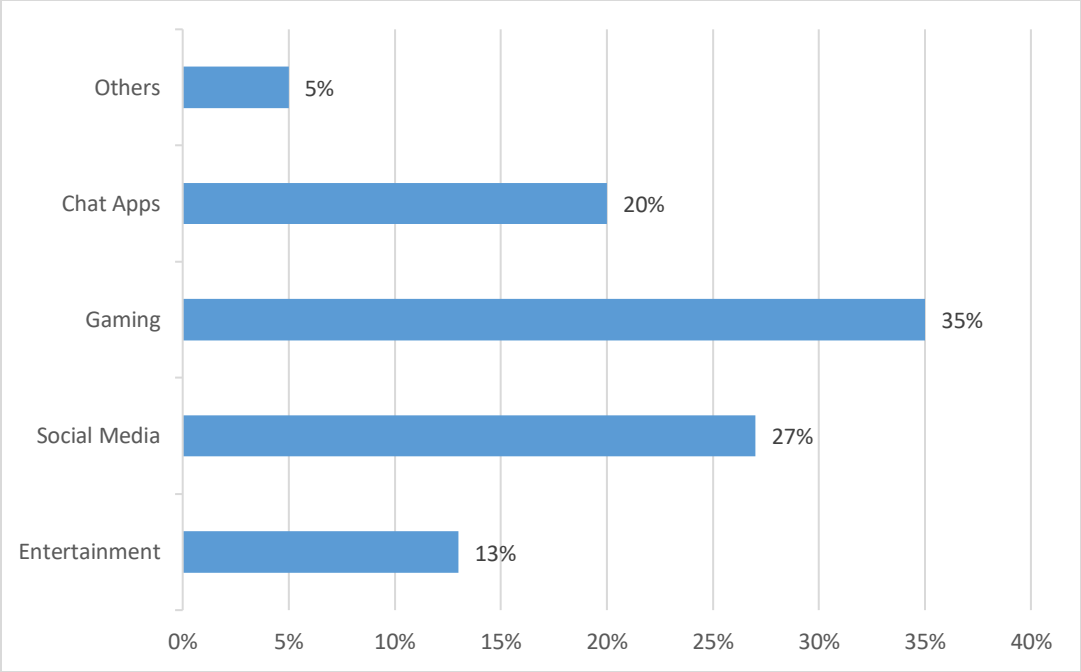


Figure 5.7 App Categories with vulnerable OpenSSL deployments

5.2.2.3 Usability Testing

The testers were asked to rate how difficult they found it to use the mobile application tool, from installation to launching of the application and performing the scans on the various applications installed on their devices. The rating levels to choose from were as follows.

- Very Easy
- Easy
- Not Hard
- Hard
- Very Hard

Their responses were as provided in the figure below.

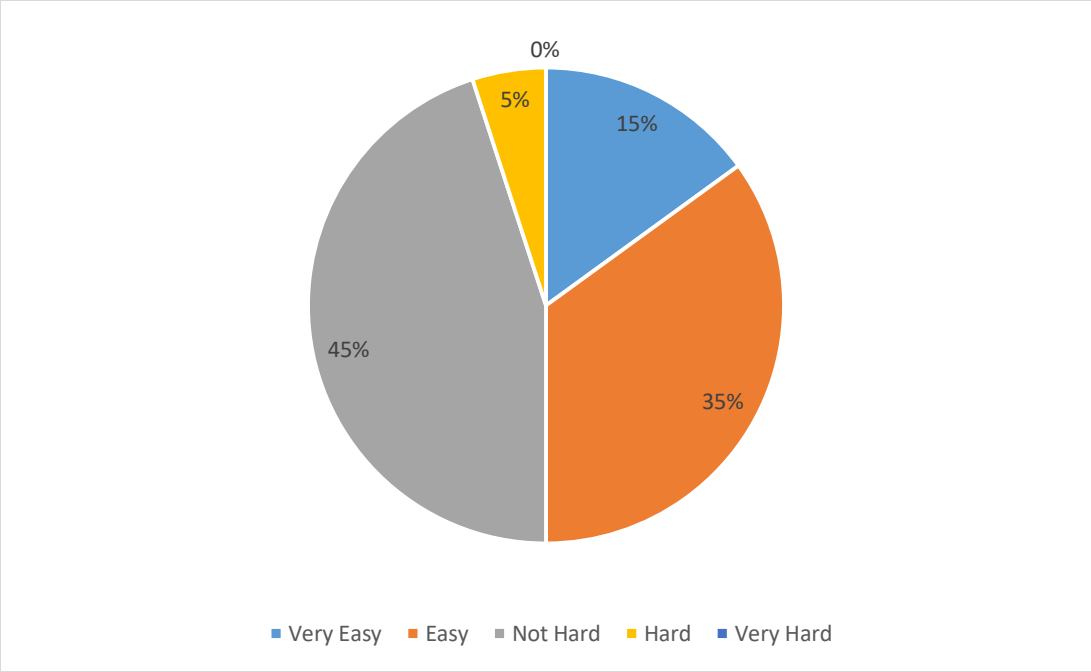


Figure 5.8 App Usability Rating

Chapter 6: Discussion of Results

6.1 Web Testing Results

The feedback received from the web testing team was mainly positive. The testers found it easy to register and manage the details of vulnerable versions of OpenSSL. Access to the online portal was also straight forward and easy to do. Navigation around the website was easy due to consistency of the navigation menu location. Testers also found the menu headings easy to understand. The testers also found the process of creating and managing of user accounts for the testers to use when accessing the web portal to be straight forward.

However, there was a problem accessing the website through mobile devices. The elements were not displayed in a responsive manner. The layout of the website did not scale well with the varying view ports of the various mobile devices that were used to access the website. The testers also discovered that once deleted, user accounts could not be reactivated again. A request to have this feature introduced was made. The website was also accessed through and insecure link that exposed the login credentials to anyone who was listening to network traffic on that address.

All in all, the testers found it easy to access the web platform and update the details of vulnerable OpenSSL versions.

6.2 Mobile Application Testing Results

All the mobile application testers were highly against having to remove the security restrictions on their devices that prevented installation of applications from unknown sources. From this one could deduce that mobile device users have a basic understanding of the need to secure their devices against vulnerabilities.

The testers were also not particularly fond of having to type out the entire URL address in order to download the mobile application. Concerns were raised over how secure the link was. Having to both download the mobile application from an unfamiliar URL link and to remove the security restrictions on their mobile devices played a huge part in the rating of the usability of the mobile application. Had the mobile application been downloadable from a proper app store such as Google Play Store, it would be safe to say that the usability ratings would have been different.

When it came to performing the scan on selected mobile applications installed on the devices, the response received from the server on vulnerable OpenSSL versions was clear, concise and very informative. This was well received with the testers. The ability to find out more on the identified vulnerability was also very well received by the testers.

Although the primary focus of this study was how secure mobile banking applications were, it was noted that gaming and social media applications made more use of OpenSSL libraries than mobile banking applications. This is because they, gaming and social media applications, offer in-app purchases or payments of various products within the application. This means that the user will have to submit their credit card information and other payment details that can be grabbed by an attacker performing a man in the middle attack. It is therefore important to have a secure deployment of an OpenSSL library within these kinds of applications.

65% of users chose to retain the application on their mobile devices well after the testing period had lapsed. From this, it was evident that this solution was well received by the testers.

Chapter 7: Conclusions, Recommendation and Future Work

7.1 Conclusion

This research's focus was to develop a tool that would scan mobile applications for vulnerable OpenSSL libraries. The tool was developed and it was able to perform its duties successfully. This success was evident in its ability to scan a selected mobile application for OpenSSL libraries that might have been deployed and perform a comparison of the detected version with those vulnerable versions registered on the web application, giving recommendations where positive matches were found. As an additional bonus, it was also able to scan the user's mobile device for OpenSSL libraries residing on it and perform a comparison with vulnerable versions registered on the portal, also giving recommendations when a positive match was found.

7.2 Recommendation

It was highly recommended that the web portal be made to be responsive to mobile devices. This would make it easier for the system administrators to access the portal from their mobile devices and update the records from there. It would also be a good idea to have a mobile application for system administrators to use when updating vulnerabilities.

Given that this solution was designed to work only on an Android driven device, there were recommendations put forward to avail the application on other operating systems as well such as Apple's iPhone and Blackberry. It was also recommended that the mobile application be uploaded to a credible app store for the relevant mobile platform. This will increase the users' confidence in the solution.

7.3 Future Work

It was evident from this study that most mobile banking applications for Kenyan banks did not make use of any OpenSSL libraries to secure their communication as anticipated. Further research will need to be conducted to investigate the security measures put in place by banks to secure communications by their mobile banking applications.

References

Acunetix. (2014, April 17). The Aftermath of the Heartbleed Bug [Network security website]. Retrieved June 9, 2017, from https://www.acunetix.com/blog/articles/aftermath-heartbleed-bug/

Brignall, M. (2016, April 16). Sim-swap fraud claims another mobile banking victim [News Website]. Retrieved April 16, 2016, from https://www.theguardian.com/money/2016/apr/16/sim-swap-fraud-mobile-banking-fraudsters

Chandra, P., Messier, M., & Viega, J. (2002). Network Security with OpenSSL. In *Network Security with OpenSSL* (p. 384). O'Reilly. Retrieved from http://directory.umm.ac.id/Networking%20Manual/Network%20Security%20With%20OpenSSL%202002.pdf

Codenomicon. (2014, April 29). Heartbleed [Informational]. Retrieved June 9, 2017, from http://heartbleed.com/

Equity Bank. (2014, May). *Equity MVNO Strategy and Roll-out Plan*. PowerPoint Presentation presented at the Investors Briefing. Retrieved from www.equitybankgroup.com/index.php/files/download/827

Equity bank's APIs are now open to developers. (2016, May 11). [news website]. Retrieved December 10, 2016, from http://www.cio.co.ke/news/top-stories/equity-bank's-apis-are-now-open-to-developers

Gangan, S. (2015). A Review of Man-in-the-Middle Attacks, 12. https://doi.org/arXiv:1504.02115

Global Black Market for stolen personal data. (2016). Retrieved January 26, 2017, from http://www.trendmicro.com/vinfo/us/security/special-report/cybercriminal-underground-economy-series/global-black-market-for-stolen-data/

Google. (2016). How to addres OpenSSL vulnerabilities in your apps. Retrieved December 14, 2016, from https://support.google.com/faqs/answer/6376725?hl=en

Guguyu, O. (2015, April 25). Internal fraud bleeds banks dry [News Website]. Retrieved December 14, 2016, from http://www.nation.co.ke/business/Internal-fraud-bleeds-banks-dry/996-3174878-mj8g4fz/index.html

Improving Time To Market. (n.d.). Retrieved January 3, 2017, from http://www.arenasolutions.com/resources/articles/time-to-market/

Jamsheer, K. (2016, August 19). 12 Best Software Development Methodologies with Pros and Cons [Information Technology Communication]. Retrieved December 27, 2016, from http://acodez.in/12-best-software-development-methodologies-pros-cons/

Mashable. (2014, April 10). The Heartbleed Hit List: The Passwords You Need to Change Right Now [Informational]. Retrieved June 9, 2017, from http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/#Wgz_cuGEyaqr

Mettler, A., Zhang, Y., & Raman, V. (2014, August 20). SSL Vulnerabilities: Who listens when Android applications talk. Retrieved December 23, 2016, from https://www.fireeye.com/blog/threat-research/2014/08/ssl-vulnerabilities-who-listens-when-android-applications-talk.html

NIST. (2014, October 14). Vulnerability Summary for CVE-2014-3566 [Information Communication Technology]. Retrieved January 3, 2017, from https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-3566

OpenSSL: Vulnerability Statistics. (2017, April 24). [Security Vulnerability Data source]. Retrieved April 24, 2017, from https://www.cvedetails.com/vendor/217/Openssl.html

Ouko, E. (2013, June). Mobile Banking Adoption in the Banking Industry on Kenya. Retrieved from http://chss.uonbi.ac.ke/sites/default/files/chss/ESTHER%20A.%20OUKO%20ODERA%20D61-71319-2008_0.pdf

Paladion Networks. (2007, May 23). Mobile Banking Architecture [Information Communication Technology]. Retrieved December 12, 2016, from http://paladion.net/mobile-banking-architecture/

PCI Security Standards Council. (2015). Migrating from SSL to Early TLS, 7.

Peck, M., Kini, G., & Pyles, A. (2016, March). Android Security Analysis Final Report. Retrieved from http://roselabs.nl/files/audit_reports/MITRE_-_Android.pdf

Porteous, D. (2006, May). The Enabling Environment for Mobile Banking in Africa. DFID. Retrieved from https://www.microfinancegateway.org/sites/default/files/mfg-en-paper-the-enabling-environment-for-mobile-banking-in-africa-may-2006_0.pdf

Rajendran, M. (2014, February 14). Security Alert: Frauds can clone your SIM, use your credit card [News Website]. Retrieved December 12, 2016, from http://www.hindustantimes.com/business/security-alert-frauds-can-clone-your-sim-use-your-credit-card/story-cnstNfrpsdE5SSDA5YjkKK.html

Rakitin, S. (1999). Balancing Time To Market and Quality. Retrieved from http://www.swqual.com/images/Balancing.pdf

Rouse, M. (2015, October). Phishing [Information Security]. Retrieved January 3, 2017, from http://searchsecurity.techtarget.com/definition/phishing

Rouse, M. (2016, February). Social Engineering [Information Security]. Retrieved January 3, 2017, from http://searchsecurity.techtarget.com/definition/social-engineering

SDLC - Waterfall Model. (n.d.). [Information Communication Technology]. Retrieved January 6, 2017, from https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

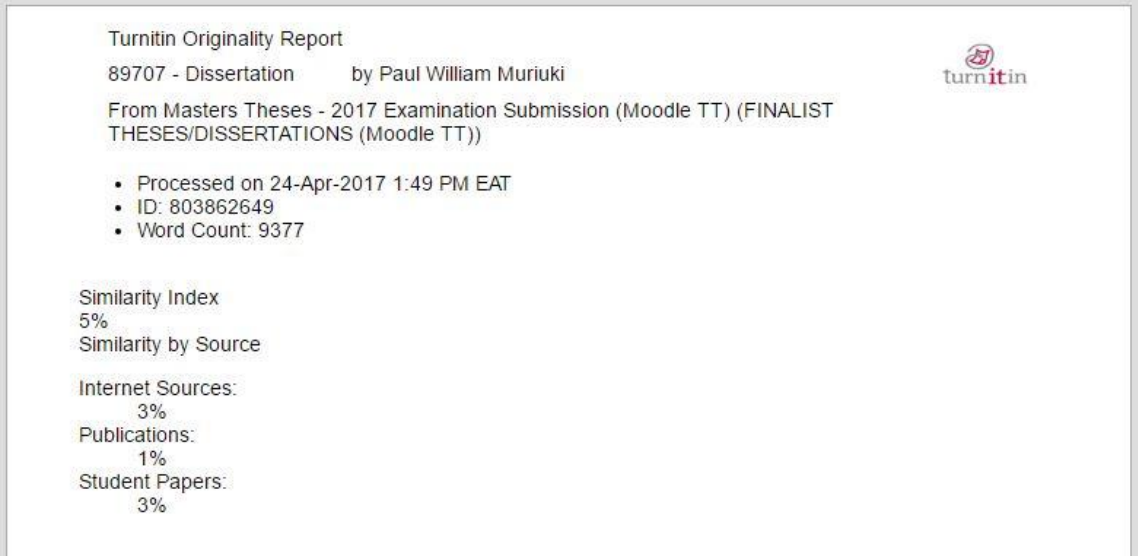
Serianu, S. (2015). Kenya Cyber Security Report 2015. Retrieved from http://serianu.com/downloads/KenyaCyberSecurityReport2015.pdf

Temenos, & NetGuardian. (2016). A-Z of Banking Fraud 2016, 39.

Waters, K. (2011, May 4). Agile Development Cycle [Software development methodologies]. Retrieved January 6, 2017, from http://www.allaboutagile.com/agile-development-cycle/

Weak Diffie-Hellman and the Logjam Attack. (2015). Retrieved December 13, 2016, from https://weakdh.org/

Appendix



Turnitin Originality Report

89707 - Dissertation by Paul William Muriuki

From Masters Theses - 2017 Examination Submission (Moodle TT) (FINALIST THESES/DISSERTATIONS (Moodle TT))

- Processed on 24-Apr-2017 1:49 PM EAT
- ID: 803862649
- Word Count: 9377

Similarity Index
5%
Similarity by Source

Internet Sources:
3%
Publications:
1%
Student Papers:
3%

Appendix A-1: Turnitin Report