



UNIVERSITY OF GHANA LEGON
ACCRA GHANA

DEPARTMENT OF COMPUTER ENGINEERING
SCHOOL OF ENGINEERING SCIENCES

FINAL YEAR PROJECT REPORT
ON

**INTEGRATED NETWORK MONITORING SYSTEM FOR THE
UNIVERSITY OF GHANA COMPUTING SYSTEMS (UGCS)**

PROJECT REPORT SUBMITTED IN PARTIAL FULLFILMENT OF THE
REQUIREMENTS FOR THE BACHELOR OF SCIENCE DEGREE IN
COMPUTER ENGINEERING

NAME OF STUDENT: **KWADWO AMO-ADDAI**

INDEX NUMBER: **10537189**

NAME OF SUPERVISOR: **DR. GODFREY A. MILLS**

NAME OF CO-SUPERVISOR: **MR. BEN COBBLAH**

DATE OF SUBMISSION: **FRIDAY, 8TH JUNE 2018**

TITLE PAGE

**INTEGRATED NETWORK MONITORING SYSTEM FOR THE
UNIVERSITY OF GHANA COMPUTING SYSTEMS (UGCS)**

By
KWADWO AMO-ADDAI
(10537189)

Submitted to the Department of Computer Engineering in
Partial Fulfilment of the Requirements for the Degree of
Bachelor of Science in Computer Engineering

University of Ghana

FRIDAY, JUNE 8TH 2018

Name of Student: KWADWO AMO-ADDAI.

Signature of Student: _____

Name of Supervisor: DR. GODFREY A. MILLS.

Signature of Supervisor: _____

Name of Head of Department: DR. ROBERT A. SOWAH.

Signature of Head of Department: _____

DECLARATION OF ORIGINALITY

Department of Computer Engineering

I certify that this thesis is my own work except where indicated by referencing. I also certify that I have read and understood the guidelines on plagiarism in the Computer Engineering undergraduate thesis handbook including the University of Ghana's policy on plagiarism. I have acknowledged in the text of this thesis all sources used. I have also fully referenced including page numbers all the relevant texts, figures, data, and tables quoted from books, journals, articles, Internet websites, and works of other people. I have not used the services of any professional person or agency to produce this document. I have also not presented the work of any student present or past from other academic or research Institutions. I understand that any false claim in respect of this thesis document will result in disciplinary action in accordance with the regulations of the University of Ghana.

Please complete the information below by Hand and in BLOCK LETTERS.

Student Name:

Index Number:

Student Signature: Date.....

Name of Supervisor:

Supervisor's Signature: Date.....

ABSTRACT

INTEGRATED NETWORK MONITORING SYSTEM FOR THE UNIVERSITY OF GHANA COMPUTING SYSTEMS (UGCS)

Network monitoring is the use of tools or systems to constantly monitor a computer network for feedback on its state and that of its components and perform some corresponding actions or notify a network administrator based on the feedback.

The University of Ghana Computing Systems (UGCS) institution currently uses different network monitoring tools like Bro, Ossec, Kismet, and Lynis to monitor the university's network. These tools however, work as independent units without integration, making it very difficult to coordinate the different monitoring operations which they perform. There is also a need for the information produced by these tools to be made more human-readable, to be easily analysed and understood by the system's end-users. And lastly, there is a need for an auto-alerting functionality within the system, where the personnel are alerted on any issues that occur within the network in real time. This project was aimed at developing an integrated platform that could harmonize all these tools and their operations, and also process their information, as well as perform remedial actions on the information obtained and auto-generate alert messages containing the actions performed for the system's end-users.

The integrated system was developed, using a revised Software Development Life Cycle (SDLC) process, taking into consideration means of performing different network monitoring operations such as "Network Connection Tracking", "Session Tracking", "Intrusion Detection", "Vulnerability Scanning", "System Auditing", etc on the university's network and retrieving the log data using the available tools, and also processing the data into useful information, as well as performing remedial actions and auto-alerting the respective end-users of the system based on real-time detection of anomalies within the information.

The system, after rigorous testing, could perform all its functional requirements successfully, but for some non-functional requirements however, like robustness and security, there seemed to be some room for improvement. All in all, the system proved to substantially reduce the workload of monitoring any network, whether simple or complex, and can therefore be used by many organizations such as networking departments, telecom companies, etc.

The system can also be expanded to monitor very large networks by simply adding more tools with even more network monitoring operations.

ACKNOWLEDGEMENT

First and Foremost, I would like to express my sincere gratitude to my supervisor Dr. Godfrey A. Mills for being a huge resource and for providing continuous support of my study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of the research and writing of this thesis. I could not have imagined having a better advisor and mentor for my study. Thank you for being helpful and understanding, and being available for needed meetings. Besides my advisor, I extend my sincere thanks to my co-supervisor, Mr. Ben Cobblah, head of IT Security Planning at the UGCS institution for added input, and also to the Chief Information Technology Officer (CITO) of the university, Mr. Lucas Yikimpa Chigabatia and the entire UGCS institution, for providing me with the necessary resources for the project and for constantly guiding me throughout the work. Last but not the least, I would like to thank my parents Mr. and Mrs. Amo-Addai for constantly supporting me throughout my life. Words fail me to express my gratitude to my beloved mother, who sacrificed her comfort for my betterment; and a special thanks to family, friends, classmates and teaching assistants for keeping up my motivation and being understanding during this thesis.

Amo-Addai Kwadwo

ID: 10537189

Department of Computer Engineering

TABLE OF CONTENTS

<i>Declaration of Originality</i>	<i>ii</i>
<i>Abstract</i>	<i>iii</i>
<i>Acknowledgement</i>	<i>iv</i>
<i>List of Tables</i>	<i>vii</i>
<i>List of Figures</i>	<i>viii</i>

Chapter 1 – Introduction

1.1. Background.....	1
1.2. Problem Definition	5
1.3. Project Objectives	6
1.4. Project Relevance	6
1.5. Thesis Outline	7

Chapter 2 – Literature Review

2.1. Introduction	9
2.2. Review of Existing Works.....	9
2.3. The Proposed Project.....	17
2.4. Project Scope.....	18

Chapter 3 – System Design and Development

3.1. Introduction.....	19
3.2. System Overview and Functions.....	19
3.3. System Requirements and Specifications	21
3.4. System Design and Development Process.....	21
3.5. System Integration and Simulation.....	36
3.6. System Development Tools and Material Requirements.....	39

Chapter 4 – System Implementation and Testing

4.1. Introduction.....	41
4.2. System Implementation.....	41
4.3. System Testing and Results.....	45
4.4. Discussion and Analysis of Results.....	57
4.5. System Performance Evaluation and Limitations.....	57

Chapter 5 – Conclusions and Recommendations

5.1.	Introduction.....	59
5.2.	Conclusions	59
5.3.	Recommendations	59
5.4.	Observations and Challenges	60

References

Appendices

Appendix A – Snapshots.....	64
Appendix A1 - Web Server (REST API) Sample Request Tests.....	64
Appendix A2 - Web (Dashboard) Application Images	65
Appendix A3 - Mobile Application Images	69
Appendix B – Code Snippets.....	71
Appendix B1 - Web Server Application Code Snippet	71
Appendix B2 - Web (Dashboard) Application Code Snippet	73
Appendix B3 – Mobile Application Code Snippet	80
Appendix B4 - Console (CLI) Application (Sleeper Program) Code Snippet...	81

LIST OF TABLES

Table 4.1 – System Performance and Stress Testing Results.....	56
--	----

LIST OF FIGURES

Figure 3.1 – System (Top-Level) Architecture.....	19
Figure 3.2 – System Block Diagram.....	24
Figure 3.3 – System Use-Cases Diagram.....	25
Figure 3.4 – System Flow Diagram.....	26
Figure 3.5 – Database Schema.....	28
Figure 3.6 – Home Page View (Web Application)	32
Figure 3.7 – Home Page View (Mobile Application)	33
Figure 3.8 – System (Top-Level) Architecture	37
Figure 3.9 – Session Tracking Test (Client applications sign-ins)	38
Figure 3.10 – Session Tracking Handler Action Results	38
Figure 3.11 – Session Tracking Auto-Alerting Results	39
Figure 4.1 – Virtual Network Setup.....	42
Figure 4.2 – Virtual Machines.....	42
Figure 4.3 – Sample Log File Formats.....	44
Figure 4.4 – GitLab Auto-DevOps CD/CI Flow.....	51

CHAPTER 1 - INTRODUCTION

1.1. Background

This chapter aims to introduce the concept and some of the operations involved with network monitoring.

1.1.1. Network Monitoring Concept

Network monitoring, as a part of network management, is the use of tools or systems to constantly monitor a computer network for feedback on its state and that of its components and also to perform some corresponding actions based on the feedback or notify a network administrator (via various communication means such as email, SMS or other alarms) in case of occurrence of outages or other issues [2]. It can also be defined as the practice of overseeing the operations of a computer network using specialized management tools [2]. A network monitoring system is basically used to ensure availability and overall performance of network devices (or hosts) and network services. Administrators are given the opportunity to monitor access, routers, slow or failing components, firewalls, client systems, core switches and server performance among other network data [1]. Without network monitoring, it can easily take several hours to locate a network error, such as identifying a router which is not working properly. With a monitoring service, the amount of downtime can be minimized and in many cases even proactively eliminated, and piling up of costs, due to unused man days, can also be reduced.

Although network monitoring can be clearly defined, it can also be seen as very broad, in that it comprises many distinct sub-operations termed as “Network Monitoring Operations” within this thesis report. Some of such sub-operations are “Network Connection Tracking” (which is what is known as “network monitoring” to the general public), “Session Tracking”, “Intrusion Detection”, “Vulnerability Scanning”, “System Auditing”, etc. For example, while an intrusion detection system monitors a network threats from outside, a network connection tracking (monitoring) system monitors the whole network for problems caused by overloaded or crashed servers or other devices within the network. Some commonly measured metrics are response, time, availability and uptime, status request failures, such as when a connection cannot be established, or it times out or the document or messages cannot be retrieved. Problems or issues like

this, when detected, usually cause a number of remedial actions or handlers to be performed or network administrators to be alerted by the monitoring system.

Some of the commonly used network monitoring tools include Bro, Ossec, Lynis, etc. Most of these tools are open source software packages and hence can be easily adapted for specific applications, depending on end-users. For example, Bro can be used to perform Intrusion Detection, while Lynis can be used for enhanced System Auditing. These specific applications are collectively called “Network Monitoring Operations”, which will be discussed in the next section of this chapter.

1.1.2. Network Monitoring Operations

These are the specific sub-operations that are typically performed in the field of network monitoring, using network monitoring tools. Some of them are “Network Connection Tracking”, “Intrusion Detection”, “Vulnerability Scanning”, “System Auditing” etc, and they will be discussed within this section.

1.1.2.1. Network Connection Tracking

Network tracking, also called Network Device Management, is the act of keeping track of moves, additions, and changes (known as MACs) to the hardware infrastructure of a network [2]. Some network tracking tools keep track of cabling MACs only. Others can keep track of all the devices connected to the network, and provide visual diagrams of the network configuration showing all the configurations the network has had since it was first put into use. A network tracking tool allows a network administrator to maintain an inventory of network devices, schedule device scans, and compile periodic reports on the status of each device. Some network tracking tools provide a wizard or Web-browser interface to make the program more user-friendly [2]. A few network tracking tools can contact each new device and determine the make and model automatically (if the administrator chooses that option). Devices can be sorted and categorized according to manufacturer, model, make, status, and other user-defined criteria.

There are a number of advantages to the use of a network tracking tool. For example, if a cable is unplugged, an alarm can be triggered in the network operations centre. Some network tracking tools allow work orders to be signed off and documentation on the

physical layer to be updated automatically. A network tracking tool can help technicians resolve hardware problems by identifying the location in the physical layer where trouble occurs. By maintaining comprehensive records of all MACs, a network tracking tool can help an enterprise conform to the provisions of the Sarbanes-Oxley Act (SOX).

1.1.2.2. Intrusion Detection

An intrusion detection system (IDS) is a device or software application that monitors a network or systems for malicious activity or policy violations [3]. Any malicious activity or violation is typically reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system combines outputs from multiple sources, and uses alarm filtering techniques to distinguish malicious activity from false alarms [3]. There is a wide spectrum of IDS, varying from antivirus software to hierarchical systems that monitor the traffic of an entire backbone network. The most common classifications are network-based intrusion detection systems (NIDS), host-based intrusion detection systems (HIDS). A system that monitors important operating system files is an example of a HIDS, while a system that analyses incoming network traffic is an example of a NIDS. Another popular kind of IDS is a wireless-based intrusion detection systems (WIDS), which deals less with the packet payload but more with strange things happening inside the wireless protocols (mostly 802.11) and functions [21]. It is also possible to classify IDS by detection approach: the most well-known variants are signature-based detection (recognizing bad patterns, such as malware) and anomaly-based detection (detecting deviations from a model of "good" traffic, which often relies on machine learning). Some IDS have the ability to respond to detected intrusions. Systems with response capabilities are typically referred to as intrusion prevention systems, which may attempt to stop an intrusion attempt but this is neither required nor expected of a monitoring system

1.1.2.3. Vulnerability Scanning

A vulnerability scanner is a computer program designed to assess computers, computer systems, networks or applications for known weaknesses [4]. In plain words, these scanners are used to discover the weak points or poorly constructed parts. It is utilized for the identification and detection of vulnerabilities relating to mis-configured assets or flawed software that resides on a network-based asset such as a firewall, router, web server, application server, etc [22]. Modern vulnerability scanners will allow for both

authenticated and unauthenticated scans to occur. Modern scanners are typically available as SaaS (Software as a Service) by providers over the internet as a web application and the amount of host information is vast. The modern vulnerability scanner has the capabilities to customize vulnerability reports, installed software, open ports, certificates and much other host information that can be queried by users to increase network security. Here are the two main types of vulnerability scans that can be performed:

- Authenticated scans allow for the scanner to directly access network based assets using remote administrative protocols such as secure shell (SSH) or remote desktop protocol (RDP) and authenticate using provided system credentials. This allows the vulnerability scanner to access low-level data, such as specific services and configuration details of the host operating system. It's then able to provide detailed and accurate information about the operating system and installed software, including configuration issues and missing security patches.
- Unauthenticated scans can result in a high number of false positives and is unable to provide detailed information about the assets operating system and installed software. This method is typically used by threat actors or security analyst trying to determine the security posture of externally accessible assets. The CIS Critical Security Controls for Effective Cyber Defence designates continuous vulnerability scanning as a critical control for effective cyber defence.

1.1.2.4. System Auditing

System Auditing, also called Information Technology Security Assessment (IT Security Assessment), is an explicit study to locate IT systems' security vulnerabilities and risks [5]. In an assessment, the assessor should have the full cooperation of the organization being assessed. The organization grants access to its facilities, provides network access, outlines detailed information about the network, etc. All parties understand that the goal is to study security and identify improvements to secure the systems. An assessment for security is potentially the most useful of all security tests.

The main goal of a system auditing, is to ensure that necessary security controls are integrated into the design and implementation of a project [5]. A properly completed system audit or security assessment should provide documentation outlining any

security gaps between a project design and approved corporate security policies. Management can address security gaps in three ways: Management can decide to cancel the project, allocate the necessary resources to correct the security gaps, or accept the risk based on an informed risk / reward analysis.

1.2. Problem Definition

The University of Ghana Computing Systems (UGCS) institution currently uses different network monitoring (open source) tools such as Bro, Ossec, Kismet, Lynis, and some others to monitor the university's network. Each of these tools however, work as independent units without integration, making it very difficult to coordinate the different monitoring operations which they perform. This raises the need for employing multiple networking analysts and employees, each with specialized knowledge on each of these tools. Due to the different knowledge backgrounds and points of view of such analysts, there may be a rise in work conflicts between them on how network monitoring operations must be executed, causing a reduction in the harmony of such operations. Therefore, an integrated network monitoring flow is required between all the various tools, where some operations undertaken by some of them can trigger or kick-start (automatically) other operations to be undertaken by others.

Moreover, there is also a need for the information (logs) produced by each of these distinct tools to be made more human-readable, or processed into a more standard format, to be easily analysed and understood by the system's end-users, the network system administrators and analysts regardless of which source or tools the information came from. This will enable the information to be efficiently understood by any average network analyst, rather than obtaining different specialists for each of the various tools used by the organisation.

Lastly, there is a need for an integration of an auto-alerting functionality within the organisation's network monitoring system, where respective personnel are alerted on any issues that occur within the network in real time, as such message information proves to be very crucial.

1.3. Project Objectives

This project was aimed at developing an integrated platform that could harmonize all these tools and their operations, and also process their information, as well as perform remedial actions on the information obtained and auto-generate alert messages containing the actions performed for the system's end-users. The project's four main objectives are as follows:

1. Design and implementing a system that can integrate the operations of the different network monitoring tools used by the UGCS Institution.
2. Process acquired log data from the monitoring tools to more human-readable information.
3. Design and implement software applications (on both Web and Mobile platforms) to as interface systems for network administrators to view and manage the information (log data) obtained by the monitoring tools.
4. Implement an Alert functionality within the system, to enable auto-alerting of system administrators on issues occurring within the network, as well as remedial actions taken to handle them, via communication avenues such as Email, SMS, or Mobile Application notification.

Other objectives may include identifying the types of threats posed to the network with their levels of emergency, and providing solutions and techniques administrators can deploy to control such threats networks, provisioning extremely useful research into constantly changing network usage patterns, and others.

1.4. Project Relevance

The main relevance of this project is to identify the best networking monitoring solution from a network system administration point of view. There are various ways in achieving this goal, but this project will focus on one which is the integration of multiple network monitoring tools and their various operations. With this integrated system ongoing, it will be able to identify the source of malpractices like viruses, movement of copyright material etc. This system will affect the security and communications systems of institutions.

This research project also seeks to identify the causes of network failures such as inappropriate traffic such as viruses, hacking exploits, etc on institutions' networks. Due

to the absence of alert and high grade network monitoring systems, inappropriate network traffic can end up causing a complete shutdown of the network, and this served as the drive towards this research project work.

With this integrated network monitoring system, I hope to simplify the job of the network administrators so they can go about their work more efficiently and with less stress and effort.

1.5. Thesis Outline

This thesis report contains information on how the project was undertaken from the problem stage to the conclusion and closing stage. It includes 5 different chapters, where each chapter contains specific information on the process of the project. Here is a complete description of each chapter within this report.

Chapter 1, Introduction, seeks to introduce the concept of network monitoring. It also clearly states the problem at hand, as well as the project objectives and its relevance. Finally, it clearly describes the outline of this thesis, stating out all the information held within each chapter of this thesis.

Chapter 2, Literature Review, describes all the amount of research made towards completing this project and thesis report. It contains the review of existing works, in terms of the mentions network monitoring tools, as well as other related integrated network monitoring systems. It also contains the project which was proposed after the research was made and its scope.

Chapter 3, System Design and Development, the overview of the system, as well as its functions and requirements are clearly described. The chapter goes on to talk about the complete process of designing and developing the system, and also how each component of the system was integrated to form an entity and how it was simulated. Finally, the chapter talks about the development tools and material requirements for developing the system.

Chapter 4, System Implementation and Testing, talks about how the integrated system was completely implemented and tested. It also continues on to discussing and analysing the testing results in order to evaluate the system's performance and determine its limitations, if any exist.

Chapter 5, Conclusion and Recommendations, is the last chapter of this report. It clearly states all the conclusions drawn from analysing the results obtained within chapter 4, and also talks about some recommendations for expanding and enhancing the integrated system. Finally, some general observations made from running the entire project are stated within this chapter.

After all these 5 chapters, the report shows all available references obtained from research and some appendices which contains some more images of the system as well as some code snippets used for its development.

CHAPTER 2 – LITERATURE REVIEW

2.1. Introduction

This chapter describes all the amount of research made towards completing this project and thesis report. It contains the review of existing works, in terms of the mentions network monitoring tools, as well as other related integrated network monitoring systems. It also contains the project which was proposed after the research was made and its scope.

2.2. Review of Existing Works

In order to obtain extensive knowledge and background in the field of network monitoring, some amount of research was required. As such, here is a literature review of the research done on the network monitoring tools and other systems.

2.2.1. Network Monitoring Open Source Tools

The network monitoring tools discussed within this section are Bro, Snort, Suricata, Kismet, Ossec, Nikto, OpenVAS, and Lynis. These tools are all open-source software, which are software programs that can easily be retrieved from the internet by download, and adapted for particular applications, without any costs incurred.

2.2.1.1. Bro

Bro, founded by Vern Paxson, is an open source, script driven, UNIX based Network Connection Tracking, and Network Intrusion Detection System (NIDS) [6]. It has the tendency for looking out for malicious traffic. This open source tool detects intrusions by first parsing network traffic then executing event oriented analysers that compare the activity with patterns seemed as malicious ones. Analysis include detection of specific attacks (signature and events) and unusual activities (anomalous). It also has support for clustering for high throughput environments and provides a ‘worker’ based architecture to utilise multiple processors. Bro’s developers recommend allocating one core for every 80 Mbps of traffic that is being analysed [6]. It also has features allowing to interact with other systems in the enterprise, send email messages, page on-call staff, or automatically terminate existing connections. Bro also works based on file hash

extraction and matching with the use of publicly available hash registers. It is important to notice that the processing per core is significantly low compared to other NIDS tools such as Snort or Suricata [6]. But Bro has built in capacity to spread the load across multiple machines via Bro cluster thereby providing greater scalability. Also, a site can adopt Bro's operation by its specialized policy language and when new site attacks are discovered, if anything is detected, Bro can generate a log entry, alert the operator in real time and also execute an operating system command, if specified. But there has been some research showing that the overhead of distributed processing slows down the performance. Thus the performance acceleration is applicable only till a certain saturation point.

2.2.1.2. Snort

Snort, like Bro, is also a free and open source Network Intrusion Prevention System (IPS) and Network Intrusion Detection System (IDS), created by Martin Roesch in 1998 [7]. Snort is now developed by Sourcefire, of which Roesch is the founder and CTO, and which has been owned by Cisco since 2013 [27]. In 2009, Snort entered InfoWorld's Open Source Hall of Fame as one of the "greatest (pieces of) open source software of all time" [7]. It really isn't very hard to use, but there are a lot of command line options to play with, and it's not always obvious which ones go together well. Snort can be configured to run in three modes:

- Sniffer mode, which simply reads the packets off of the network and displays them for you in a continuous stream on the console (screen).
- Packet Logger mode, which logs the packets to disk.
- Network Intrusion Detection System (NIDS) mode, which performs detection and analysis on network traffic. This is the most complex and configurable mode, and actually the main mode in which it will be used within this integrated system.

2.2.1.3. Suricata

Suricata is an open source-based intrusion detection system (IDS) and intrusion prevention system (IPS), and also capable of network security monitoring (NSM) and offline pcap processing [8]. It was developed by the Open Information Security Foundation (OISF), a non-profit foundation committed to ensuring Suricata's development and sustained success as an open source project [28]. A beta version was

released in December 2009, with the first standard release following in July 2010. Suricata inspects the network traffic using a powerful and extensive rules and signature language, and has powerful Lua scripting support for detection of complex threats [8]. With standard input and output formats like YAML and JSON integrations with tools like existing SIEMs, Splunk, Logstash/Elasticsearch, Kibana, and other database become effortless. Suricata's fast paced community driven development focuses on security, usability and efficiency.

2.2.1.4. Kismet

Kismet is the baseline for Wireless Intrusion Detection System (WIDS). Wireless IDS deals less with the packet payload but more with strange things happening inside the wireless protocols (mostly 802.11) and functions [9]. WIDS will find unauthorized Access Points (**Rogue AP Detection**), perhaps one accidentally created by an employee that opens a network up [29]. Perhaps someone has stood up an Access Point (AP) with the same name as your corporate network to perform Man-In-The-Middle attacks? Kismet will find all of these. Kismet runs on a variety of platforms, even Android. Besides Intrusion Detection, Kismet can also be used for more utilitarian purposes like wireless site surveys or some other fun activities like 'WarDriving', which is a hacking tactic that involves moving around an area, whilst catching the IP Addresses and other information of wireless access points (Wi-Fi's) within that area [9].

2.2.1.4. Ossec

In the realm of full featured Open Source Host Intrusion Detection System (HIDS) tools, there is OSSEC and not much else, mainly due to its extensibility. OSSEC will run on almost any major operating system and uses a Client/Server based architecture which is very important in a HIDS system [10]. Since a HIDS could be potentially compromised at the same time the OS is, it's very important that security and forensic information leave the host and be stored elsewhere as soon as possible to avoid any kind of tampering or obfuscation that would prevent detection. OSSEC's architecture design incorporates this strategy by delivering alerts and logs to its own OSSEC centralized server where analysis and notification can occur even if the host system is taken offline or compromised [30]. Another advantage of this architecture is the ability to **centrally manage** agents from a single server. Since deployments can range from one to

thousands of installations, the ability to make changes en masse via a central server is critical for an administrator's sanity [10].

2.2.1.6. Nikto

Nikto, developed by the CFO of Open Security Foundation (Chris Sullo) for vulnerability assessment, is an Open Source (GPL) web server scanner which performs comprehensive tests against web servers for multiple items, including over 6700 potentially dangerous files/CGIs, checks for outdated versions of over 1250 servers, and version specific problems on over 270 servers [11]. It also checks for server configuration items such as the presence of multiple index files, HTTP server options, and will attempt to identify installed web servers and software. It can also scan for outdated server software, perform generic and server type specific checks, capture and print any cookies received, and scan for other miscellaneous problems [12]. Its plugins are also frequently updated and can be automatically updated (if desired). Nikto is not designed as an overly stealthy tool. It will test a web server in the shortest timespan possible, and it's fairly obvious in log files. However, there is support for LibWhisker's anti-IDS methods in case you want to give it a try (or test your IDS system) [11].

There are some variations of Nikto, one of which is MacNikto. MacNikto is an AppleScript GUI shell script wrapper built in Apple's Xcode and Interface Builder, released under the terms of the GPL [12]. It provides easy access to a subset of the features available in the Open Source, command-line driven Nikto web security scanner, installed along with the MacNikto application.

Not every check is a security problem, though most are. There are some items that are "info only" type checks that look for items that may not have a security flaw, but the webmaster or security engineer may not know are present on the server. These items are usually marked appropriately in the information printed. There are also some checks for unknown items which have been seen scanned for in log files.

2.2.1.7. OpenVAS

OpenVAS (Open Vulnerability Assessment System, originally known as GNessUs) is a software framework of several services and tools offering vulnerability scanning and vulnerability management solutions [12]. The framework is part of Greenbone

Networks' commercial vulnerability management solution from which developments are contributed to the Open Source community since 2009, and also a member project of Software in the Public Interest [11]. All OpenVAS products are free software, and most components are licensed under the GNU General Public License (GPL). Plugins for OpenVAS are written in the Nessus Attack Scripting Language, NASL [12]. OpenVAS began under the name of GNessus, as a fork of the previously open source Nessus scanning tool, after its developers Tenable Network Security changed it to a proprietary (closed source) license in October 2005 [12]. OpenVAS was originally proposed by pen-testers at Portcullis Computer Security and then announced by Tim Brown on Slashdot.

2.2.1.8. Lynis

Lynis is an open source security System Auditing tool used by system administrators, security professionals, and auditors, to evaluate the security defences of their Linux and other Unix-based systems (FreeBSD, Mac OS, OpenBSD, Solaris, etc) [13]. It runs on the host itself, so it performs more extensive security scans than does simple vulnerability scanners like Nikto and Open VAS. It assists system administrators and security professionals with scanning a system and its security defenses, with the final goal being system hardening. The tool was created by Michael Boelen, the original author of rkhunter as well as several special contributors and translators [31]. Lynis is available under the GPLv3 license. The software determines various system information, such as the specific OS type, kernel parameters, authentication and accounting mechanism, installed packages, installed services, network configuration, logging and monitoring (e.g. syslog-ng), cryptography (e.g. SSL/TLS certificates) and installed malware scanners (e.g. ClamAV or rkhunter) [13]. Additionally, it will check the system for configuration errors and security issues. By request of the auditor, those checks may conform to international standards such as ISO 27001, PCI-DSS 3.2 and HIPAA. The software also helps with fully automated or semi-automatic auditing, software patch management, evaluation of server hardening guidelines and vulnerability/malware scanning of Unix-based systems [31]. It can be locally installed from most system repositories, or directly started from disk, including USB stick, CD or DVD. Its main intended audience is auditors, security specialists, penetration testers, and sometimes system/network administrators. Usually members of a First Line of Defence within a company or larger organization tend to employ such audit tools. According to the official documentation, there is also a Lynis Enterprise version,

available with support for more than 10 computer systems, providing malware scanning, intrusion detection and additional guidance for auditors [13].

2.2.2. Integrated Network Monitoring Systems

In this section, some integrated systems related with network monitoring will be discussed as part of the literature review for this project. These related works are Cacti (cacti.net, 2015), Icinga (Icinga, 2015), Nagios (Nagios, 2009-2015), and AlienVault, which is currently the main integrated network monitoring system used by the university.

2.2.2.1. AlienVault

AlienVault is a developer of commercial and open source solutions to manage cyber-attacks, including the Open Threat Exchange, the world's largest crowd-sourced computer-security platform [14]. In July 2017, the platform had 65,000 participants who contributed more than 14 million threat indicators daily. The company has raised \$116 million since it was founded in 2007.

In addition to their free products, AlienVault offers a paid security platform, called Unified Security Management, that integrates threat detection, incident response, and compliance management into one solution [23]. Threat applications are offered via hardware, virtual machines, and as a cloud service.

The Open Threat Exchange (OTX), which is free, enables security experts to research and collaborate on new threats, better compare data and integrate threat information into their security systems [14]. A big data platform, OTX leverages natural language processing and machine learning.[4]

AlienVault also runs the Open Source Security Information Management (OSSIM) project, which helps network administrators with computer security, intrusion detection, and response [23]. The OSSIM project began in 2003 and was started by Dominique Karg, Julio Casal and later Alberto Román [23]. It became the basis of AlienVault, founded in 2007 in Madrid, Spain.

AlienVault Open Threat Exchange had 26,000 participants in 140 countries reporting more than one million potential threats daily, as of June 2015, and in February 2017, AlienVault released USM Anywhere, a SaaS security monitoring platform designed to centralize threat detection, incident response and compliance management of cloud, hybrid cloud, and on-premises environments from a cloud-based console [14]. By July 2017, AlienVault Open Threat Exchange platform had 65,000 participants who contributed more than 14 million threat indicators daily.

The main issue with AlienVault is its limited functionality. It only focuses on intrusion detection, and cannot perform other network monitoring operations like vulnerability scanning, system auditing, etc.

2.2.2.2. Cacti

Cacti is an open-source, web-based network monitoring and graphing tool designed as a front-end application for the open-source, industry-standard data logging tool RRDtool [15]. Cacti allows a user to poll services at predetermined intervals and graph the resulting data. It is generally used to graph time-series data of metrics such as CPU load and network bandwidth utilization [24]. A common usage is to monitor network traffic by polling a network switch or router interface via Simple Network Management Protocol (SNMP).

The front end can handle multiple users, each with their own graph sets, so it is sometimes used by web hosting providers (especially dedicated server, virtual private server, and colocation providers) to display bandwidth statistics for their customers [15]. It can be used to configure the data collection itself, allowing certain setups to be monitored without any manual configuration of RRDtool. Cacti can be extended to monitor any source via shell scripts and executables. Cacti can use one of two back ends: "cmd.php", a PHP script suitable for smaller installations, or "Spine" (formerly Cactid), a C-based poller which can scale to thousands of hosts [24].

The Cacti project was first started by Ian Berry on September 2, 2001. Berry was inspired to start the project while working for a small ISP while also still in high school, learning PHP and MySQL. His central aim in creating Cacti "was to offer more ease of use than RRDtool and more flexibility than MRTG" [15].

On September 13, 2004, version 0.8.6 was released, and with it came more developers and, later on, greater program speed and scalability. Version 0.8.7 was also released for use in October 2007 [24]. In June 2012, a roadmap on the website indicated that version 1.0.0 was scheduled for release in the first quarter of 2013, but by August 2013 it was replaced by a notice that "This roadmap is out of date. We are reorganizing to improve the development cycle of Cacti. More information coming before the end of 2013!" [15].

2.2.2.3. Icinga

Icinga is an open source computer system and network monitoring application. It was originally created as a fork of the Nagios system monitoring application in 2009 [16]. Icinga is attempting to get past perceived short-comings in Nagios' development process, as well as adding new features such as a modern Web 2.0 style user interface, additional database connectors (for MySQL, Oracle, and PostgreSQL), and a REST API that lets administrators integrate numerous extensions without complicated modification of the Icinga core [25]. The Icinga developers also seek to reflect community needs more closely and to integrate patches more quickly. The first stable version, 1.0, was released in December 2009, and the version counter had risen every couple of months as of January 2010. Icinga was included by Jeffrey Hammond of Forrester Research in a list of "waxing" (as opposed to "waning") open source projects based on its rate of commits [16]. The name Icinga is a Zulu word meaning "it looks for", "it browses" or "it examines" and is pronounced with a click consonant.

In May 2009, a group of developers from the Nagios community announced the fork Icinga, citing their dissatisfaction with the stagnant development of the Nagios software at the time and their desire to open its development to a broader base [25]. In their first year, Icinga developers released separate versions of Core, API and Web, and celebrated their 10,000th download.

Due to its nature as a fork, Icinga offers Nagios' features with some additions such as optional reporting module with improved SLA accuracy, additional database connectors for PostgreSQL and Oracle, and distributed systems for redundant monitoring [16]. Icinga also maintains configuration and plug-in compatibility with Nagios, facilitating migration between the two monitoring software.

2.2.2.4. Nagios

Nagios, now known as Nagios Core, is a free and open source computer-software application that monitors systems, networks and infrastructure [17]. Nagios offers monitoring and alerting services for servers, switches, applications and services. It alerts users when things go wrong and alerts them a second time when the problem has been resolved. Ethan Galstad and a group of developers originally wrote Nagios as NetSaint. As of 2015 they actively maintain both the official and unofficial plugins [17]. Nagios is a recursive acronym: "Nagios Ain't Gonna Insist On Sainthood" – "sainthood" makes reference to the original name NetSaint, which changed in response to a legal challenge by owners of a similar trademark [26]. "Agios" (or "hagios") also transliterates the Greek word ἅγιος, which means "saint" [26].

Nagios was originally designed to run under Linux, but it also runs well on other Unix variants. It is free software licensed under the terms of the GNU General Public License version 2 as published by the Free Software Foundation [25].

The main issue with Nagios is its inability to efficiently scale up to handle larger and more complex networks.

2.3. The Proposed Project

The vast growth and modification of network systems in organisations, makes it infeasible for a single individual to construct and maintain a mental frame of the entire work. A network monitoring system should endeavour to monitor the occurrences on the network without placing unnecessary load on the devices being monitored. It should also be able to perform different network monitoring operations (such as intrusion detection, vulnerability scanning, etc) concurrently in order to enhance its performance.

This I believe has been absent in the fore-mentioned network monitoring systems and can be corrected by the implementation of the proposed integrated network monitoring system, which takes away the pressure of analysing the occurrences within the network from the hosts within the network.

Also, with the network monitoring systems one should be able to receive notifications, via the various avenues mentioned (Mobile application notification, SMS, or email), on

occurrences detected by the network's event systems that can check if some form of value is within a given threshold or detect undesirable trends or anomalies within the network.

To conclude, a complete integrated network monitoring system should comprise all necessary functionalities which include integration of network monitoring operations, real-time network data retrieval, enhanced data processing into understandable information, to be easily comprehended by its end-users, event detection, alerting, and others. These functions performed by the system greatly reduce the workload of the organization using it, and which is why such an integrated system is what is being proposed for this project.

2.4. Project Scope

The proposed project, the integrated network monitoring system, only covers some of the network security techniques involved with network monitoring. These techniques or operations are "Network Connection Tracking", "Session Tracking", "Intrusion Detection", "Vulnerability Scanning", and "System Auditing". However, it does not involve other network security techniques such as firewall management, anti-virus scanning, etc. Furthermore, it also does not involve cyber-security techniques concerned with the data security and physical security categories, which may include cryptography (encryption, tokenization, anonymization, etc), data-loss prevention, access-control or door-lock systems, etc.

CHAPTER 3 – SYSTEM DESIGN AND DEVELOPMENT

3.1. Introduction

In this chapter, the overview of the integrated network monitoring system, as well as its functions and requirements are clearly described. The chapter goes on to talk about the complete process of designing and developing the system, and also how each component of the system was integrated to form an entity and how it was simulated. Finally, the chapter talks about the development tools and material requirements for developing the system.

3.2. System Overview and Functions

The integrated network monitoring system comprises a number of sub-systems or sub-components, each with different specific functions. The top-level architecture of the integrated network monitoring system is shown in Fig. 3.1.

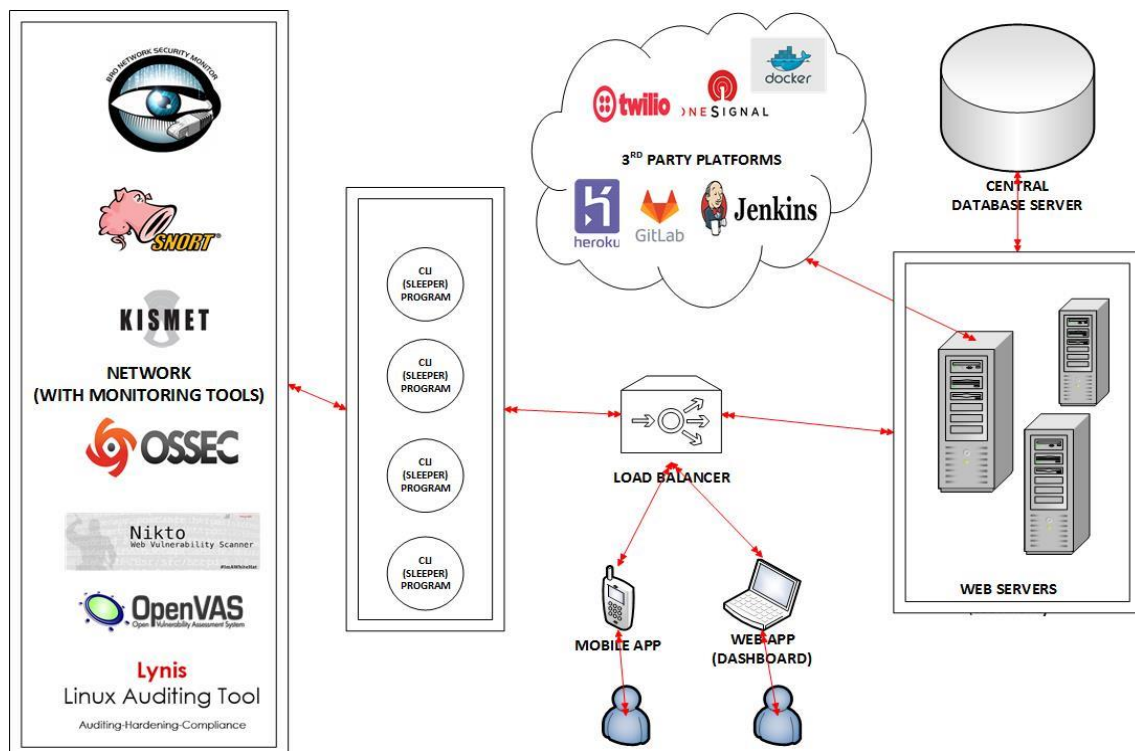


Figure 3.1 – System (Top-Level) Architecture

One can see the interconnection between all the various the sub-components within the proposed system; the client applications with the load balancer and the cluster of worker

web servers, the central database and the 3rd party tools, and the sleeper programs distributed within the entire network, inside some special chosen host machines on which all the network monitoring open-source software tools have been installed and therefore can be accessed. Before continuing, we must discuss the functionalities of all the individual components present within the integrated system. They are:

- A Centralised Shared Database Server for data (User, Log, and Event data) and file (system settings configuration) persistence
- A number of Worker Web Servers for managing the central database
- A Client Web Application acting as a dashboard for system administrators
- A Client Mobile Application for other end-users
- A number of Command Line Interface (CLI) Applications (sleeper programs) distributed throughout the network (depending on the network's size and complexity), in order to perform the network monitoring operations using the different network monitoring open-source tools (already discussed during the previous chapter), with the web servers acting as their coordinators (or supervisors, if you may)
- A Load Balancer for diversifying web requests coming from the client and CLI (sleeper programs) applications and the within the network
- 3rd Party platforms (Cloud Web Services) to aid in the alerting functionalities and other miscellaneous operations like Code and Dependency Management, DevOps, Continuous Delivery, Integration, Testing, and Deployment, etc
- Lastly, the network itself (whether physical or virtual) on which the system will reside to perform the network monitoring operations

Unfortunately, due to limited time and resources, some components may not be obtained. Some of these components may include the Load Balancer, a large number of worker web servers, etc. However, to improvise, some alternatives can be employed to fit the specific need and scope of this project. For example, for the worker web servers, only 2 can be obtained by prediction (1 being a localhost machine owned by the project leader himself, and another being a cloud host machine provided by a free cloud server hosting platform as a service).

3.3. System Requirements and Specifications

This section outlines all the requirements and specifications considered for the design and development of the integrated system. The functional requirements can be summarized as follows:

1. System must be able to integrate 2 or more of the network monitoring operations
2. System must be able to integrate 2 or more of the open source tools
3. System must be able to effectively process all available log information from all tools it integrates into a more human-readable format
4. System must be able to detect anomalies within the monitored network, then perform some actions, as specified by its administrators
5. System must be made available to all its users on both web and mobile interface platforms
6. System must allow its users to be able to successfully manage its data (its log data and user profile information)
7. System, whenever required, must be able to push alerts to its users both manually, by its administrators, and automatically, on information retrieved from (or anomalies detected within) the network

The non-functional requirements can be summarized as follows:

1. System must be robust, even in harsh conditions
2. System must be secure in terms of the CIA Triad (Confidentiality, Integrity, and Availability)
3. System must function in real time
4. System must be as flexible and human user-friendly as possible
5. System's material requirements must not be too costly

3.4. System Design and Development Process

The main focus was to design, implement, and unit test all the sub-components of this integrated system separately, before they were integrated and tested as a whole. The main approach employed for developing the system is what we Software Architects like to call the “Software Development Life Cycle” (SDLC). SDLC, a term in Software Architecture, Software Development, and Software Engineering, is a process followed in a software project, within a software or technology-oriented organization [19]. It consists of a detailed plan describing how to develop, maintain, replace and alter or

enhance specific software [19]. It defines a methodology for improving the quality of software and the overall development process [20]. SDLC is a process used by IT analysts in order to develop or redesign high quality software system, which meets both the customer and the real-world requirement. It takes into consideration all the associated aspects of software testing, analysis and post-process maintenance. It entails 6 important phases namely: Planning, Defining, Designing, Building, Testing, and Deployment. The order of these 6 stages may intertwine based on the type of Software Development and Engineering techniques chosen for the project in question.

This section describes the decision making process for deriving the system's software architecture, as well as the individual stages in which each sub-component was developed.

3.4.1. Software Architecture and Development Decision-Making

Within this project, a number of software architectural patterns and anti-patterns were employed and avoided (respectively). For the sake of clarity, a software architectural pattern is a principle (high-level decision) employed repeatedly within the development of a software system that yields positive results, whereas an anti-pattern is that which yields negative results. Also, some software development (low-level) and integration decisions were made to guide the system's development and integration process

Main Software Architectural Patterns employed:

- Traditional Layered Architecture - Persistence (Database) Layer, Services (Web Server) Layer, Business (Client Applications) Layer
- Event-Driven Architecture - Independent components' operations and processes pushing out events to others' operations and processes
- Service Oriented Architecture - Independent components provide services to themselves using a common messaging and integration pattern

Main Software Architectural Anti-Patterns avoided:

- Architecture by Implication – Individual components lacking a clear and documented architecture
- Spider Web Architecture – Addition of a large number of components which are never actually used or required by the end-users

Main Integration Patterns employed:

- Shared Database – Central Database that can be accessed by multiple Web Servers (Workers) in the case of expansion (scaling up or scaling out) of the system
- Remote Procedure Calls (RPCs) – Integration of individual components' operations and processes via RESTful web services and protocols such as HTTP
- Messaging between Decoupled Components – Asynchronous messaging between of individual components using a common data format (JSON), or and Integration Hub, in the case of expansion

Main Software Development Patterns employed:

- Agile Technique – Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).³ It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.
- SCRUM – Scrum is an agile framework for managing work with an emphasis on software development. It is designed for teams of 3 to 9 developers who break their work into actions that can be completed within time-boxed iterations, called sprints (30 days or less, most commonly two weeks) and track progress and re-plan in 15-minute stand-up meetings, called daily scrums. Approaches to coordinating the work of multiple scrum teams in larger organizations include Large-scale Scrum (LeSS), Scaled Agile Framework (SAFe) and scrum of scrums, among others.

3.4.2. ATAM (Architecture Trade-offs Analysis Method)

ATAM is a software architectural technique which takes in a proposed architecture, business drivers and system requirements as input, performs analysis in order to provide a revised and approved architecture as output. It involves main stages namely: Partnership and Preparation, Evaluation Phase 1, Evaluation Phase 2, and Follow Up stages. In software architecture, any proposed software architecture must completely go through the entire ATAM process, where it is analysed and compared with other possible architectures to evaluate its advantages and disadvantages (trade-offs), in order to derive a final revised architecture that seems to be the optimal option for the available

business drivers and system requirements and specification. As a professional software architect, I run multiple different architectures through the ATAM process before arriving at a final architecture, which will be discussed next.

3.4.3. Final Revised System Architecture

Fig. 3.2 shows the block diagram of the integrated system, containing all sub-components, each with some of its functionalities.

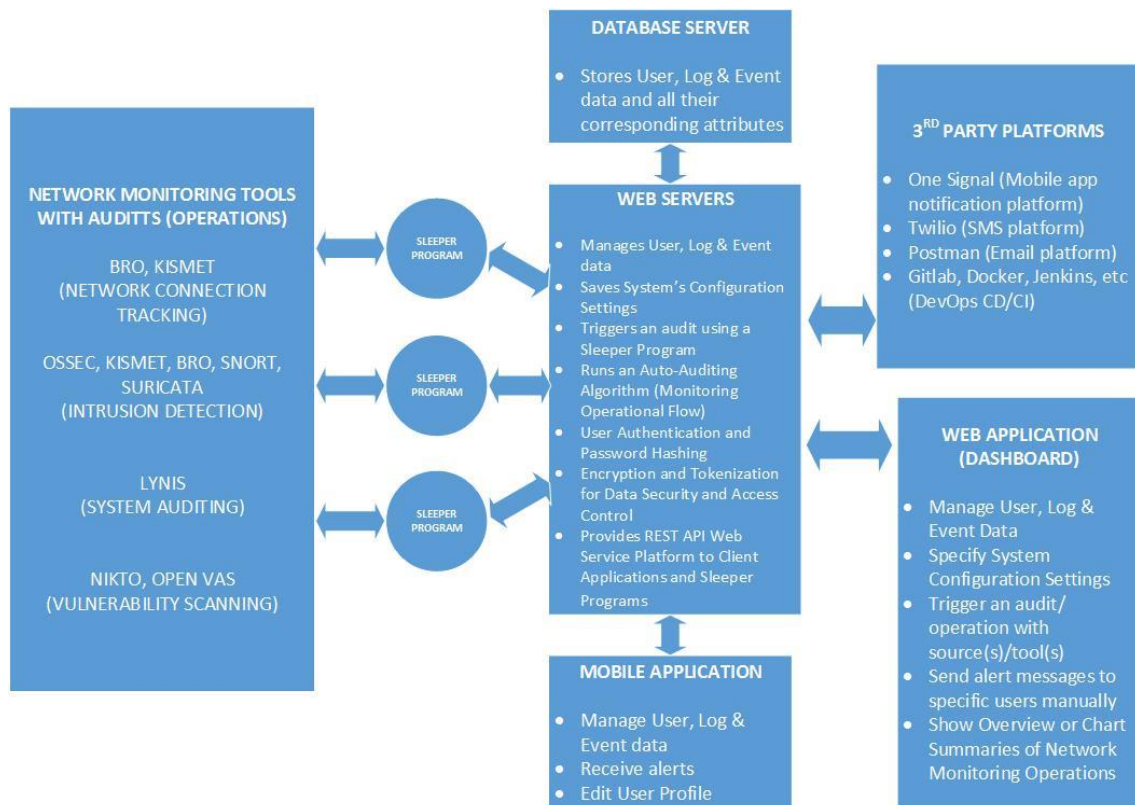


Figure 3.2 - System Block Diagram

By estimation, due to the very few number of developers (only 1), this project should take at most 36 weeks to reach completion. Here is a more descriptive project scope and schedule:

- System Architecture Design: 2-3 weeks
- Database and Web Servers Setup and Development: 6-7 weeks
- Web (Dashboard) Application: 5-6 weeks
- Mobile Application Development: 4-5 weeks
- Console (CLI) application development: 1-2 weeks

- Network and Monitoring tools installation: 2-3 weeks
- Unit Testing and Integration Testing: 4-5 weeks
- Performance, Stress, and Security Testing: 4-5 weeks

3.4.4. UML Diagrams and Notations

This section provides more information of the integrated system, using some UML (Unified Markup Language) Diagrams and Notations. Fig. 3.3 shows the main use-cases of the system, and Fig. 3.4 shows the main flow of the integrated network monitoring process of the system.

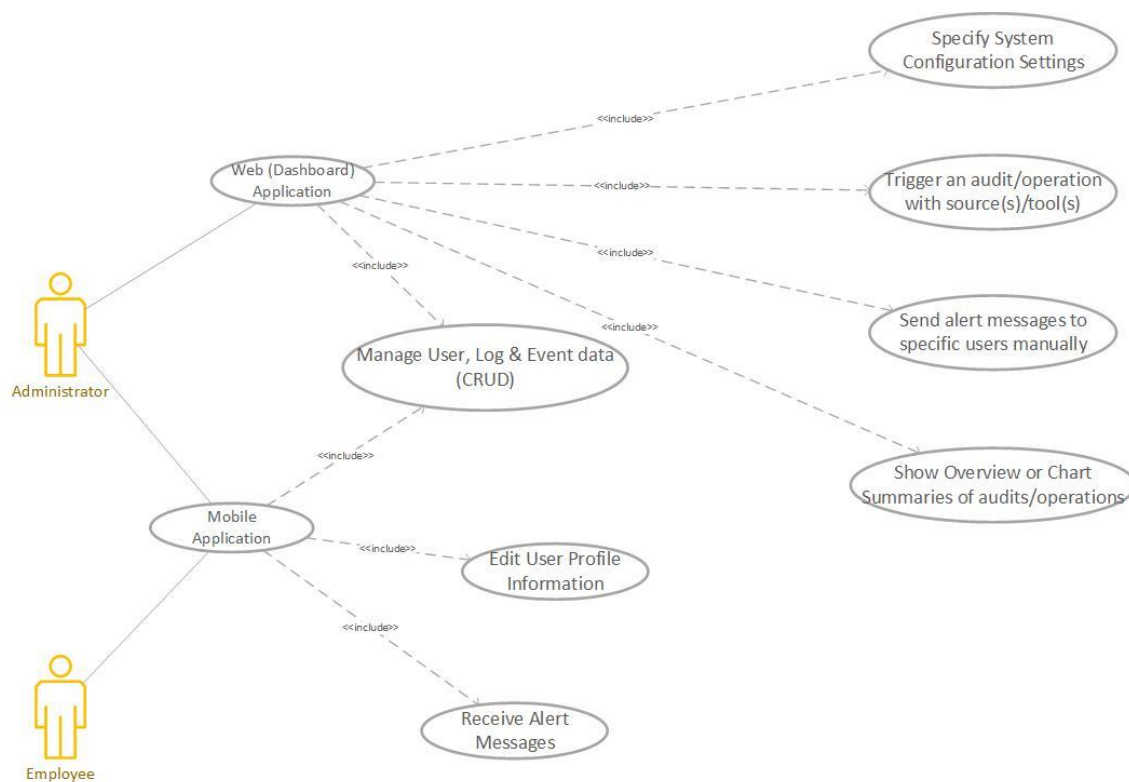


Figure 3.3 - System Use-Cases Diagram

This Use-Case Diagram above clearly states all the possible uses or functions from which the end-users (administrators and employees) can choose, using either the client web application, the “Dashboard”, or the client mobile application, currently compatible with the only the Android mobile Operating System platform.

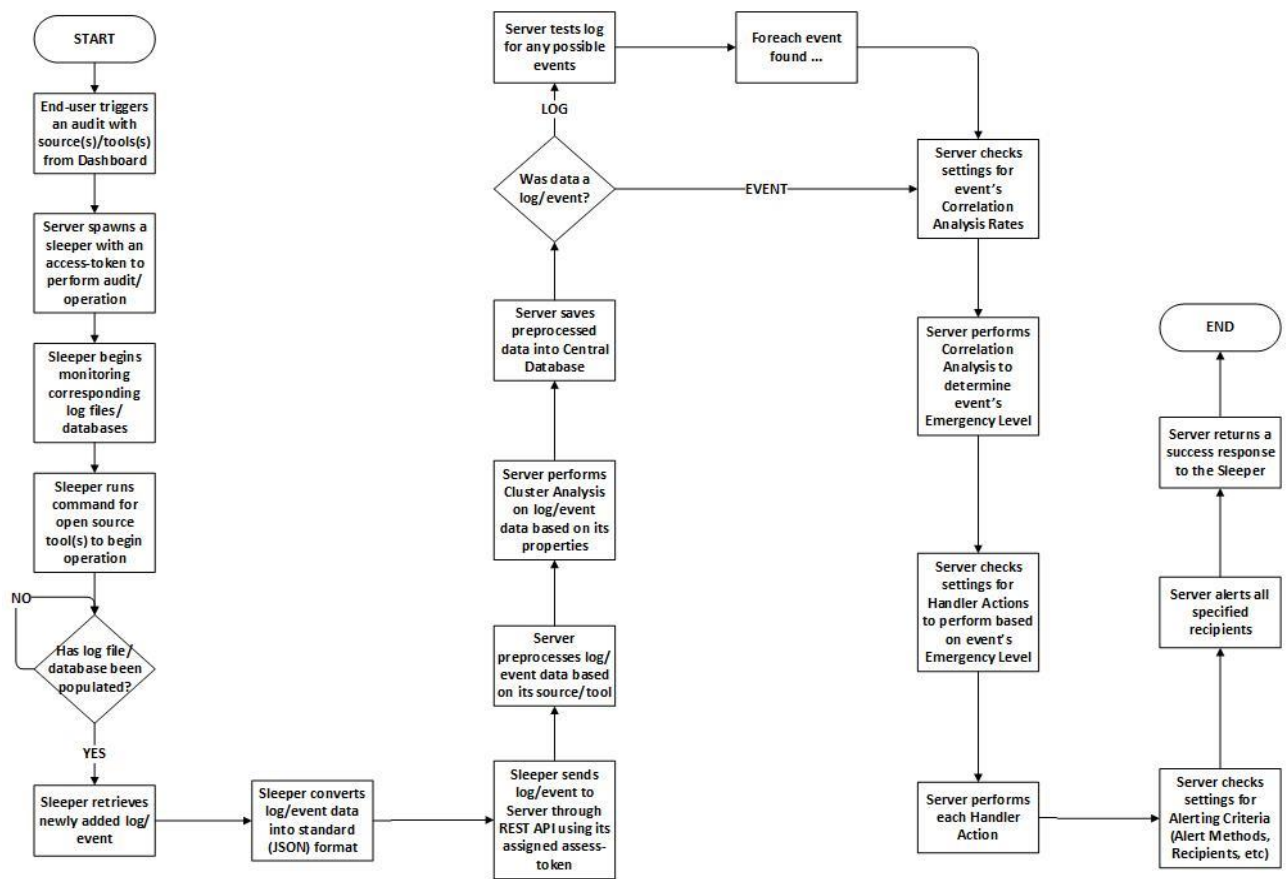


Figure 3.4 - System Flow Diagram

This System-Flow Diagram depicts the operational flow that any worker web server undergoes in order to perform all the necessary functions required, when an end-user triggers a Network Monitoring Operation using the Dashboard web application. For the sake of flexibility and ease-of-use, this operational flow can also be manipulated by the system administrators through **Configuration Settings Specification**, that can also be performed using the Dashboard. Some of the main configuration settings that can be specified are as follows:

- Log Analysis methods to perform, whether just Cluster Analysis, or with Correlation Analysis too
- Correlation Rates, to be used by the Correlation Analysis Engine to determine any given event's emergency level (whether low, medium, or high). For example, one can specify the system to treat a given event as **low** emergency level if that event occurs 10 times within 15 minutes, or **medium** emergency level if that event occurs 10 times within 10 minutes, or **high** emergency level if that event occurs 10 times within 5 minutes.

- Alerting Criteria, whether the alert message should be auto-generated, or whether a custom alert message should be sent, which recipients (users) to send the alert message to, and also on which emergency level the event must be to even send the alert message in the first place, all on the occurrence of all possible events for all possible audits (network monitoring operations)
- Event Handler Actions to perform, which can be specified for the occurrence of all possible events for all possible audits (network monitoring operations), based on the event's emergency level

NB: These Event Handler Actions are also particular audits or operations that can be performed by their corresponding open-source tools. Therefore, one notices that these actions being performed automatically by the server on specification creates an automated and integrated network monitoring flow within the system (for example, the system administrator can specify within the configuration settings that an “intrusion detected” event/anomaly detected by Bro while performing the “Intrusion Detection” audit/operation can cause the web server to automatically trigger a “Vulnerability Scanning” audit/operation to be performed by Nikto)

3.4.5. Software Development Life-Cycle (SDLC) – Sprints

The entire agile development (SCRUM) process was divided into 10 sprints or iterations, and they are:

- **SPRINT 1** – Database Server Design and Setup
- **SPRINT 2** – Web Server (and REST API Web Service) Setup and Development
- **SPRINT 3** – Web Application (Dashboard) Development
- **SPRINT 4** – Mobile Application Development
- **SPRINT 5** – 3rd Party (Cloud Web Service & DevOps) Platforms Setup and Integration
- **SPRINT 6** – Console (CLI) Application (Sleeper Program) Development
- **SPRINT 7** – System Integration and Simulation
- **SPRINT 8** – Network (Physical and Virtual) Setup
- **SPRINT 9** – Tools Installation and Log file (Datasets) Gathering
- **SPRINT 10** – Web Server, Client and Console Applications Installation

3.4.5.1. Database Server Design and Setup

For the development of the prototype, 2 databases were developed, both using the MongoDB Database framework and uri format “mongodb://username:password@host:port/database”, and they are:

1. Locally setup database on the personal computer of the project leader with uri “mongodb://localhost/projectapi-dev”
2. Live setup database using the MongoLab (mLab) MongoDB database platform with uri “mongodb://kwadwoamoad:PROJECT@ds233238.mlab.com:33238/projectapi”

Both databases included 3 data domains, **User**, **Log**, and **Event**, and their data schema diagrams are shown in Fig.3.5.

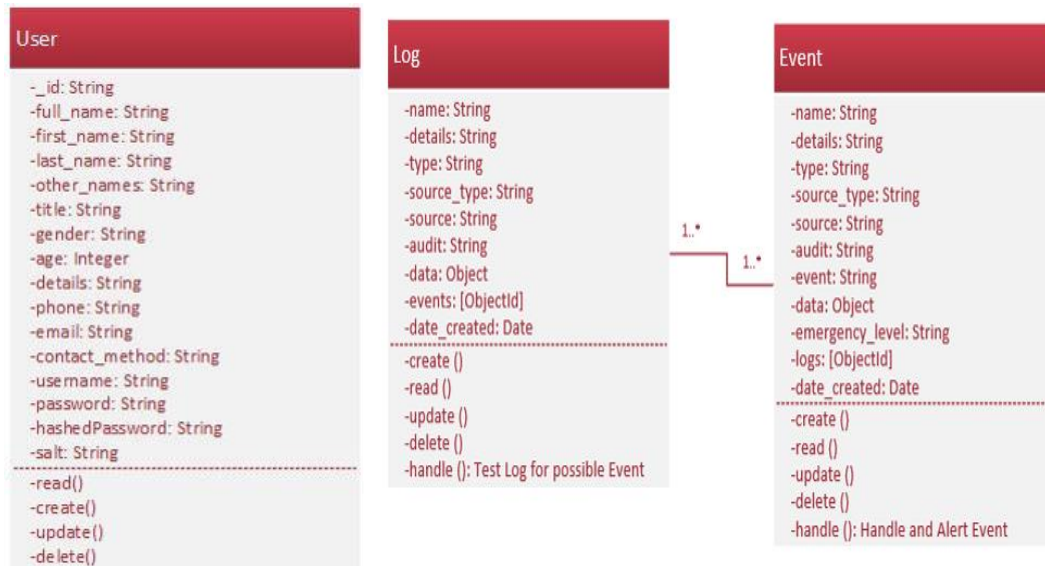


Figure 3.5 – Database Schema

The Central Database’s main function is to act as a Data Warehouse, housing and storing User, Log & Event data and all their corresponding attributes. Looking at each of these database schemas, one can see that each data domain object can be easily managed by CRUD (Create, Read, Update, Delete). Taking the **User** Schema, any specific user’s properties like “phone” and “email” aid in directing alert messages to him/her, or “contact_method” which is the user’s own personal preference as an alert method (whether SMS, E-mail, or Mobile Application Notification).

Also, taking the **Log** and **Event** Schemas, attributes like “source” (the open-source software tool/source which provided the corresponding log/event), “audit” (the network monitoring operation being performed when the log/event was detected), “date_created” (the exact timestamp of the log/event) “data” (the log/event data in its raw standard format - JSON), “name” & “details” (summary sentence representations of the “data” attribute which are way more human-readable), etc can be found. However, for the **Event** Schema alone, one can find attributes like “event” (a unique title assigned to the particular event found while performing a particular audit/operation), and “emergency_level” (the level of emergency of the corresponding event – whether low/medium/high as described earlier).

3.4.5.2. Web Server (and REST API Web Service) Setup and Development

The Web Servers are programmed server application scripts which have been setup and installed on a number of server host machines within the network, which can be called Worker Servers. These worker servers share all the network traffic load amongst themselves, with the help of an Elastic Load Balancer, which can be used to scale the worker servers up/out and in/out. Each worker web server within the integrated network has the following main functions:

- Manages User, Log & Event data with CRUD, and other operations such as sending users custom alert messages, etc
- Saves System’s Configuration Settings specified by the system’s administrators, using the Dashboard Web Application
- Triggers an audit/operation using a Sleeper Program
- Runs an **Auto-Auditing** Algorithm (Monitoring Operational Flow)
- User Authentication and Password Hashing
- Encryption and Tokenization for Data Security and Access Control
- Provides REST API Web Service Platform to Client Applications and Sleeper Programs

The Auto-Auditing Algorithm

This algorithm was divided into 3 main steps and they are:

1. Gathering, Pre-processing, and Clustering of log data (based on source tool) to be stored in database
2. Handling Log Data – Log Analysis (Testing for Possible Events)
3. Handling Event Data (with specified Handler Actions, and Alerting Criteria)

This algorithm is a fully-generic process, which each worker web server can perform on any incoming log/event object through the server's REST API Web Service. It starts from when any log/event object is received by the server (coming from any open-source tool/source through the CLI Sleeper Program), then its data is pre-processed based on the source it came from.

After that, the log/event goes through **Cluster Analysis**, where every incoming log/event is categorized according to its specific properties or attributes like the **source** (tool) from which it came, and the **audit** (operation) being performed during which it was detected, before the log/event is saved within the Central Database based on its assigned category. Note that this process, Cluster Analysis, just aims to make the querying of specific segments and categories of logs/events much easier, by preventing the querying of unwanted logs/events from the Central Database.

At this point, if the incoming data was a log the algorithm continues on to test that log data object for any of possible events (anomalies) concerned with the log's corresponding audit/operation, and then the algorithm continues on to perform more functions within the process, using all the events derived from the log. However, if it was already an event, then algorithm just continues straight away with the process.

The Web Server now performs **Correlation Analysis** on the incoming event, where its timestamp is compared with that of previously saved events under the same category/cluster it belongs to, in order to determine its emergency level, before being updated within the Central Database again, with its newly assigned emergency level. Note that an event's emergency level helps the Server know what actions to perform and whether or not to alert the systems end-users on occurrence of that particular event.

Now, that the current event has been assigned its new emergency level, the Server now moves on to checking the system's configuration settings in order to know what particular **Handler Actions** to perform based on the event's emergency level. Note that these Event Handler Actions are also particular audits or operations that can be performed by their corresponding open-source tools (through the CLI Sleeper Programs of course). Therefore, one notices that these actions being performed automatically by

the server on specification creates an automated and integrated network monitoring flow within the system.

Finally, the Web Server now checks the system's configuration settings again for the alerting criteria, whether the alert message should be auto-generated, or whether a custom alert message should be sent (which can also be specified within the configuration settings), which recipients (users) to send the alert message to, and also on which emergency level the event must be to even send the alert message in the first place.

After performing all these functions, the algorithm ends for that particular incoming log/event. However, because of the Handler Actions also automatically triggering new audits/operations to be performed by their various sources/tools (through the CLI Sleeper Program again), one realizes that those Handler Actions also detect new logs/events which are also pre-processed by the Sleeper Program and sent to the Web Server to kick-start this entire **Auto-Auditing** Algorithm again, and hence inducing an **Integrated Network Monitoring Flow** between all the available open-source tools/sources, which happens to be the main requirement/specification for this system.

3.4.5.3. Web Application (Dashboard) Development

The Dashboard web application serves as an avenue for system administrators to monitor the entire overview of the integrated network monitoring system in real time. It has the following main functions:

- Manage User, Log & Event Data by performing CRUD operations on them
- Specify System Configuration Settings (or set default configurations) through a user-friendly Graphical User Interface
- Trigger an audit/operation with source(s)/tool(s)
- Send alert messages to specific users manually
- Show Overview or Chart Summaries of Network Monitoring Operations

The next figure Fig. 3.6 shows the home page or interface of the web application, and more images of the web application can be viewed within the appendices.

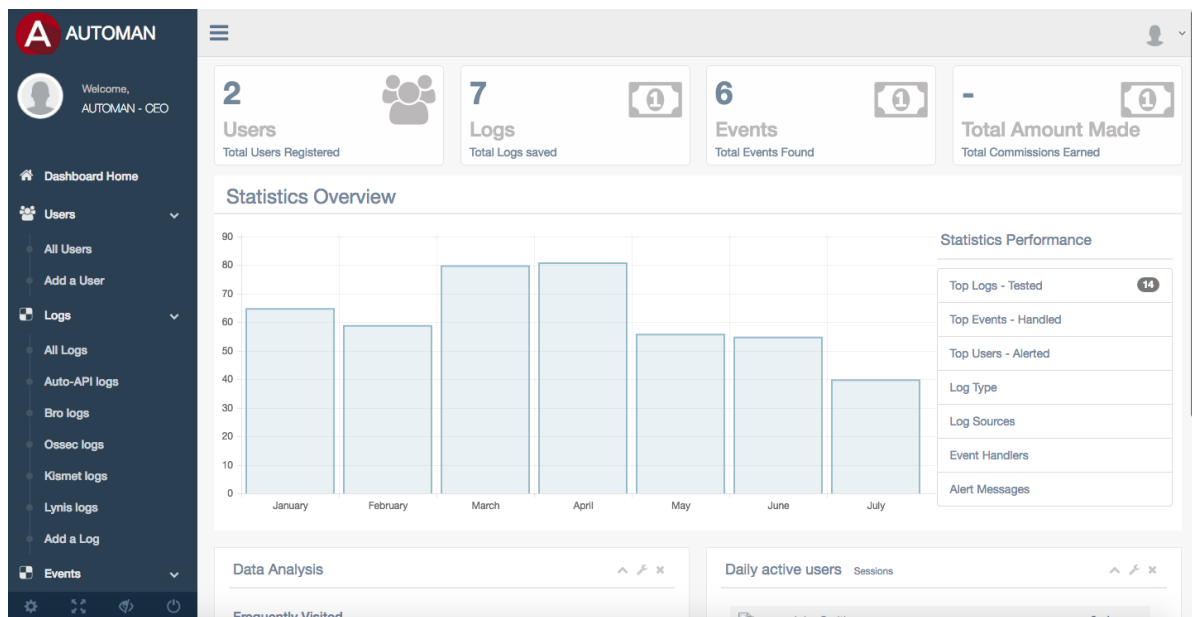


Figure 3.6 - Home Page View (Web Application)

3.4.5.4. Mobile Application Development

The mobile application serves as an avenue for the organization's employees or users (network analysts) to monitor the entire overview of the integrated network monitoring system in real time, update their user profile information, and also receive alerts through SMS, E-mail, or mobile application notification. It should be developed in compatibility with all mobile Operating System platforms, but for the sake of limited time and resources, it was developed for only the Android OS. It has the following main functions:

- Manage User, Log & Event Data by performing CRUD operations on them
- Receive alerts from the system
- Edit User Profile Information

Fig. 3.7 shows the home page or interface of the mobile application, and more images of the mobile application can be viewed within the appendices.

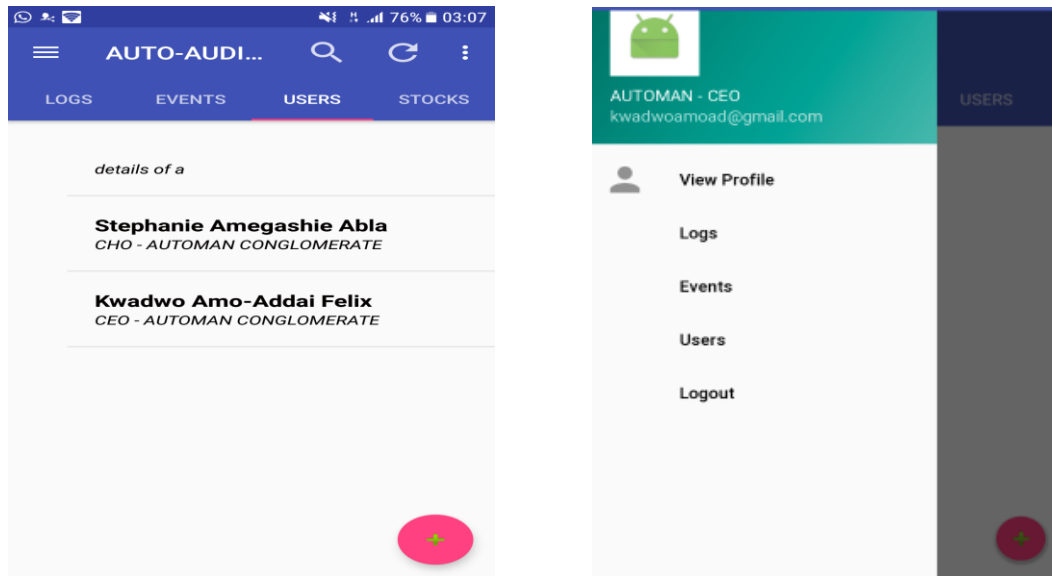


Figure 3.7 - Home Page View (Mobile Application)

3.4.5.5. 3rd Party (Cloud Web Service & DevOps) Platforms Setup and Integration

The system included some external or 3rd party platforms seamlessly integrated with its architecture. These platforms mainly served the purposes of providing **Cloud Web Services** and **Development Operations (DevOps) Services**. Cloud Web Services include all services on the world wide web that follow the principles of cloud computer architecture, being in the form of a Platform As A Service (**PaaS**), Software As A Service (**SaaS**), or Infrastructure As A Service (**IaaS**). However, DevOps Services is a subset of Cloud Web Services that mainly focus on service platforms used to integrate the “Software Development” and “Operations” (Computer Systems Engineering) sides of a Software Engineering project; they can aid in operations such automated and continuous compilation, delivery, testing, integration, and deployment of any software undergoing the development process.

The Cloud Web Service Platforms integrated within the system include:

- **OneSignal** – Platform for providing mobile application notification and device management
- **Twilio** – Platform for SMS messaging service functionality
- **Postman Email** – NodeJS code dependency for emailing functionality
- **Heroku** – Cloud Server hosting services (with Amazon Web Services – AWS)

The DevOps Service Platforms integrated within the system include:

- **Grunt** – Precompiler for the Web Server Scripts; used for automating compiling, testing, minifying, and packaging the Web Server application
- **Gulp** – Precompiler for the Web Application Scripts; used for automating compiling, testing, minifying, and packaging the Web (Dashboard) application
- **GitLab** – Cloud-based Version Control Service (VCS) for Large Code-Base Management; used for housing and protecting projects' resources
- **Docker** – Server package container; multi-platform compatible (Linux, Windows, MacOS, etc); compiles server into an image file called a “Docker Image”, then packages image file within a “Docker Container”, which can run on almost every available Operating System runtime
- **Jenkins** – Platform for automating code deployment to VCS (GitLab)
- **GitLab Auto-DevOps** – DevOps Sub-feature of GitLab platform for automating Continuous Delivery and Integration (CD/CI), and Continuous Deployment of the software system

3.4.5.6. Console (CLI) Application (Sleeper Program) Development

A sleeper program is a Command-Line Interface (CLI) utility program which runs in the background of a host machine without an end-user seeing it in action. It works with more privileges than does a normal application software program because of its utilitarian nature. The sleeper program is installed and stored within some particular host machines within the network on which the open-source network monitoring tools have been stored (because the sleeper program will need to access them to perform the audits/operations). It is triggered by a Web Server through shell commands, if the web server resides on the same host machine as the sleeper program, or by RPC (ssh/http/udp/tcp/etc) if the web server resides on a host machine remotely from the host on which the sleeper program resides.

Multiple instances of the sleeper program can be spawned or triggered by web server. However, any given instance of the sleeper program can only perform 1 audit/operation, but with 1 or more sources/tools (network monitoring open-source tools). For example, a web server can trigger or spawn an instance of the sleeper program to perform “Intrusion Detection”, using 2 network monitoring tools/sources, perhaps Bro and Ossec.

Here are the steps of operations of any instance of the sleeper program:

- Web Server spawns the instance through shell command or RPC, then assigns it an access token (so it can communicate back with the web server with the log/event data), and other parameters such as the audit/operation to run, and the corresponding network monitoring tools to be used to run that audit/operation.
- Sleeper begins to monitor the log/event data stores of each source; data stores may be special databases, log files, console output, etc

NB: Only retrieving log/event data from the log files data store has been implemented for now. The other avenues mentioned (database, console output, etc) will be implemented later, during expansion of the system

- After monitoring the data stores (log files), the Sleeper executes all the sources specified by the Web Server to run the audit/operation specified
- Now, as the sources/tools perform the audit/operation, they populate their data stores (log files), which happened to be monitored by the Sleeper, therefore the Sleeper retrieves the newly added data being added in real-time
- Sleeper pre-processes all newly added data to standard (JSON) format, because the data, coming from different sources/tools will be in different formats
- After pre-processing, the Sleeper sends the log/event data to the Web Server which spawned it, using the access token it was assigned
- The Web Server runs the **Auto-Auditing Algorithm** once it receives the log/event data from the Sleeper Program

Python Code Library Dependency Requirements:

- “ASYNC-IO” Library for executing shell commands concurrently (executing operations of network monitoring tools)
- “Watchdog” File System Monitoring Library
- “PANDAS” Big Data Analytics Library (Not yet obtained)
- “Requests” Library to make HTTP requests to Central Server

Audits/Operations, and their corresponding sources:

- Network Connection Tracking – Bro, Kismet
- Intrusion Detection – Bro, Kismet, Ossec
- System Auditing – Lynis
- Vulnerability Scanning – Nikto, OpenVAS

Log files to be monitored:

- BRO (Network Connection Tracking) – conn.log, http.log, ssh.log, ftp.log, smtp.log
- BRO (Network Intrusion Detection) - notice.log, signatures.log
- SNORT (Network Intrusion Detection) – alert.fast, alert.full
- NIKTO (Vulnerability Scanning) – custom_file_name.csv, custom_file_name.xml
- OSSEC (Host Intrusion Detection) – alerts.log, alerts.json
- KISMET (Wireless Intrusion Detection) – kismet_xml.log, dhcp.log, dns.log
- LYNIS – (System Auditing) lynis.log

Shell program (Open-source tools/sources) default commands:

- BRO – “bro <ip_address_of_network>”
- SNORT – “snort -c snort.conf -A fast -h <ip_address_of_network >”
- OSSEC – “/var/ossec/bin/ossec-control start”
- KISMET – “kismet <ip_address_of_network >”
- NIKTO – “nikto -h <ip_address_or_host_name_of_network > -o <custom_file_name.csv>”
- LYNIS – “lynis audit system”

For now, the Sleeper program just runs these default commands of the open-source tools. However, in the case of expansion of the project, it could be made to run them with different commands and with different parameters, based on some specifications that the system administrator specifies using the Dashboard web application, when triggering an audit/operation.

3.5. System Integration and Simulation

At this point, there are 3 SDLC sprints left to discuss; however, they will be discussed in the next chapter, under the section “System Implementation”, because they are mainly concerned with setting up the actual network and integrating it with this integrated network monitoring system, to be used for live implementation and testing. Before moving on, let us again view the system top-level architecture of the integrated network monitoring system in Fig. 3.8.

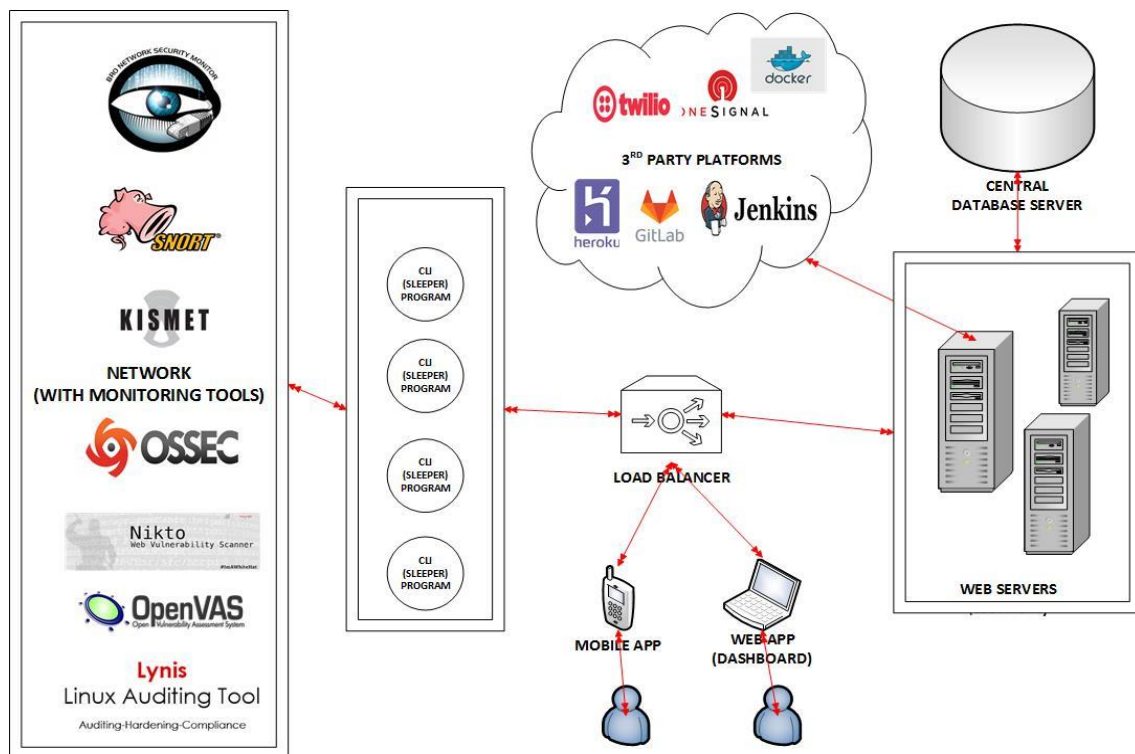


Figure 3.8 – System (Top-Level) Architecture

At this juncture, the development of all sub-components of the system have been fully described. Now, it is time to discuss how all sub-components were integrated to work together. In order to do this, we must perform a sample test using the operations of all the sub-components to work in integration. However, it was decided not to perform tests on any of the network monitoring operations/audits already discussed within this report (once again they are: “Network Connection Tracking”, “Intrusion Detection”, “Vulnerability Scanning”, and “System Auditing”), as the actual sample (virtual) network had not been setup at the time. Instead, a new audit/operation, called “**Session Tracking**” was formulated and implemented within the system.

The “Session Tracking” audit/operation deals with monitoring user sign-ins into the system. Only 1 possible event (anomaly), “**User Multiple Logins**”, was defined for this audit. It is an event used to describe when a user tries to sign into the system multiple times with different devices (without signing out in-between the sign-ins). For this “User Multiple Logins” event, the possible Event Handler Actions which were defined are “**Blacklist User Access Token**” (which means putting the user’s access token used to access the system’s network in a certain blacklist, in order to deny the user access)

and **“Delete User”** (which basically means deleting the user from the system’s Central Database).

Therefore, testing scenario was formulated, where a user tries to sign-in to the system, both on the mobile and web applications, without signing out of the system, causing the system to detect a **“User Multiple Logins”** event, and performing the Event Handler Actions specified within the system’s configuration settings, through the dashboard. The results are as follows:

Signing-in on both Web and Mobile applications without in-between sign-outs can be viewed in Fig. 3.9.

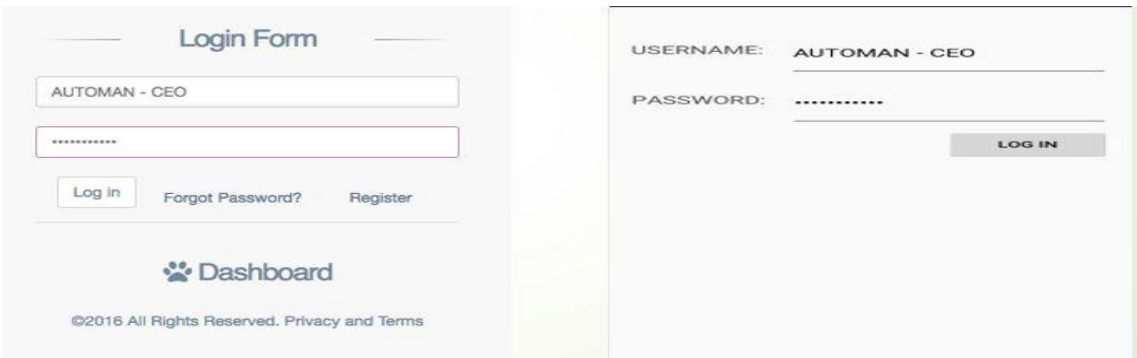


Figure 3.9 – Session Tracking Test (Client applications sign-ins)

System performs handler action (based on settings – **“Blacklists user’s access token”**), which can be seen in Fig. 3.10.

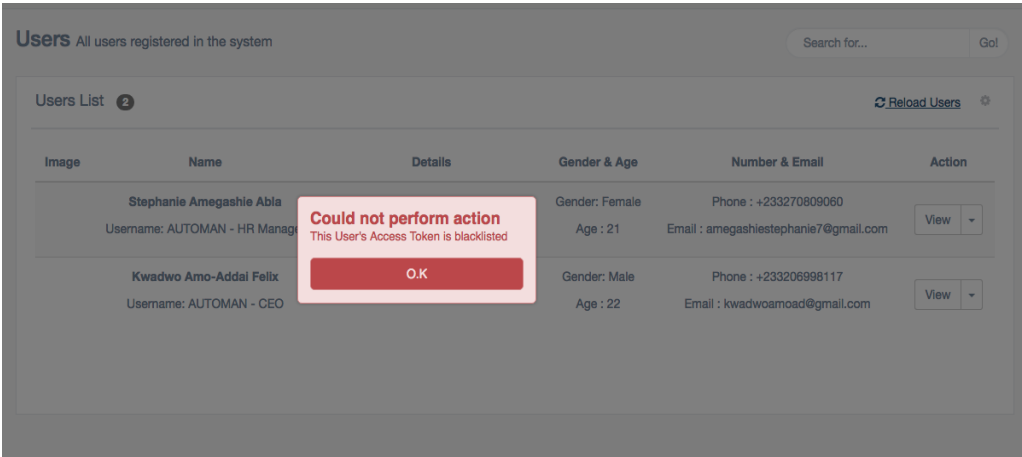


Figure 3.10 – Session Tracking Handler Action Results

System finally auto-alerts recipients (based on settings), as seen in Fig. 3.11.

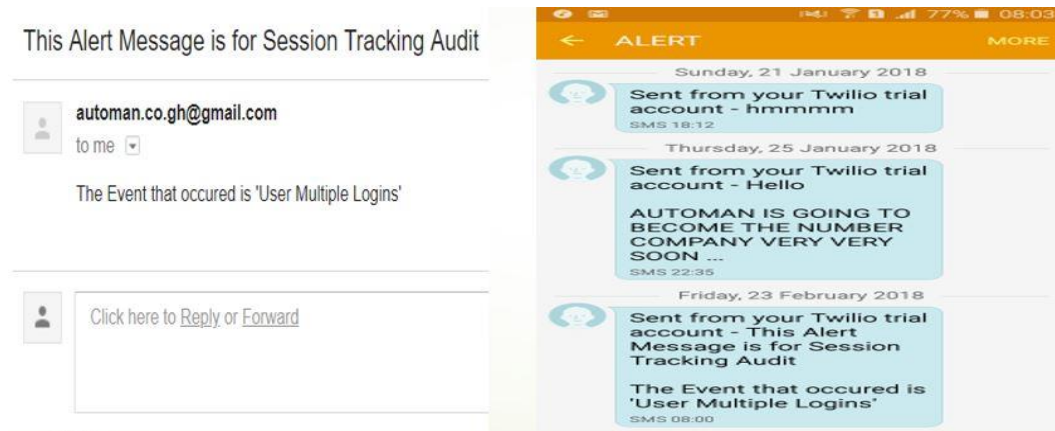


Figure 3.11 – Session Tracking Auto-Alerting Results

3.6. System Development Tools and Material Requirements

The Frameworks and tools (can be found in the system architecture diagram) to be used to develop this system are as follows:

- MongoDB for the Database Server
- NodeJS Express Framework for the Web Servers with REST API
- One Signal (Notification), Twilio (SMS), and Postman (Email) 3rd Party Cloud Services for the Alerting functionality
- AngularJS and JQuery for the Dashboard Web Application
- Grunt and Gulp as automated task runners for Web Servers and Web Application
- Docker, Jenkins, Ansible, Gitlab, MongoLab, and Heroku 3rd Party Cloud Services for Code Management and DevOps (Continuous Delivery, Integration, Testing, and Deployment of system)
- Android SDK and Java for the (Android) Mobile Application
- Python, with Pandas and Watchdog frameworks for the CLI (Sleeper) program
- Virtual Box for setting up a sample Virtual Network Environment
- Network Monitoring Tools to be integrated: Bro (NIDS), Snort (NIDS), Kismet (WIDS), Ossec (HIDS), Nikto (Vulnerability Scanning), Lynis (Linux System Auditing), which will be provided by the University of Ghana Computing Systems (UGCS)

- Kali Linux OS tools (nmap, aircrack, metasploit, etc) for Penetration Testing of the integrated network system

For the tools and frameworks mentioned above, there will be no incurred cost, in the case of developing the prototype. However, during the system's expansion and integration with the UGCS's live network, some costs, such as that for acquiring networking host machines to house the Database and Web Servers, will be sprung up.

CHAPTER 4 – SYSTEM IMPLEMENTATION AND TESTING

4.1. Introduction

In this chapter, how the integrated system was completely implemented and tested will be discussed. We will also continue on to discussing and analysing the testing results in order to evaluate the system's performance and determine its limitations, if any exist.

4.2. System Implementation

At this point, the last 3 sprints within the SDLC will be discussed, as they involve the setting up of the actual network to be integrated with the network monitoring system, for implementation and testing. The last 3 sprints are as follows:

- **SPRINT 8** – Network (Physical and Virtual) Setup
- **SPRINT 9** – Tools Installation and Log file (Datasets) Gathering
- **SPRINT 10** – Web Server, Client and Console Applications Installation

4.2.1. Network (Physical and Virtual) Setup

There cannot be a network monitoring system without a network of host machines. Therefore, a sample network, which would go through monitoring, was to be setup. During development, 2 networks were setup; 1, a virtual network, by the project leader (myself), and another, a live physical network, by the University of Ghana – UGCS Institution. Both networks were successfully setup, but only the virtual network was used to test the system, because the live network was not setup early enough to undergo testing.

Virtual Network:

Virtual Box was the tools used to setup the virtual network. The network included 2 Virtual Machines (Linux Ubuntu and Windows 10 operating systems) as nodes connected to the personal laptop host machine owned by the project leader. This laptop was the same localhost on which a Web Server, with the Sleeper Program were installed. Here is a list of the individual hosts within the network, and their corresponding IP Addresses:

- Personal laptop host – Virtual Box Host-Only Network (192.168.1.1)
- Linux Ubuntu Virtual Machine – 192.168.1.3

- Windows 10 Virtual Machine – 192.168.1.5
- Kali Linux Virtual Machine – this VM was not included within the network. This is because, it's a penetration (security) testing Operating System, so it was intended to be used for security testing of the network, after being integrated within the entire network monitoring system

In Fig. 4.1, the setup of the virtual network is shown, and in Fig. 4.2, the interfaces of the Virtual Machines within the network is also shown.

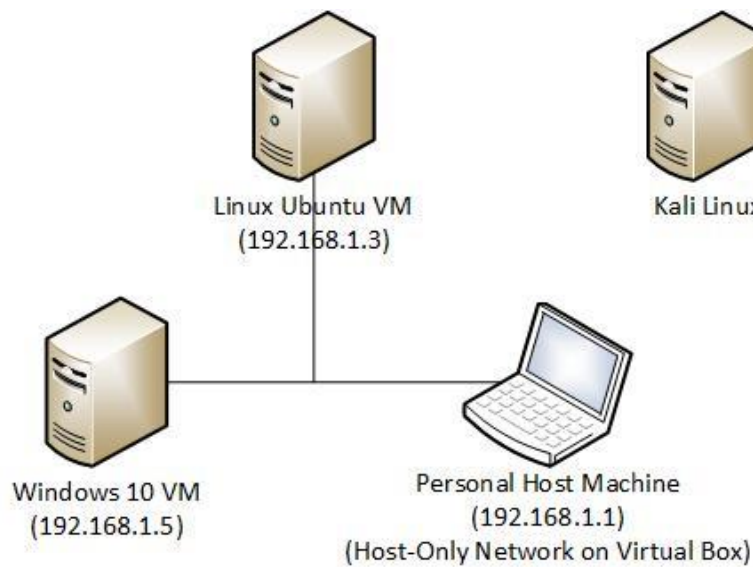


Figure 4.1 – Virtual Network Setup

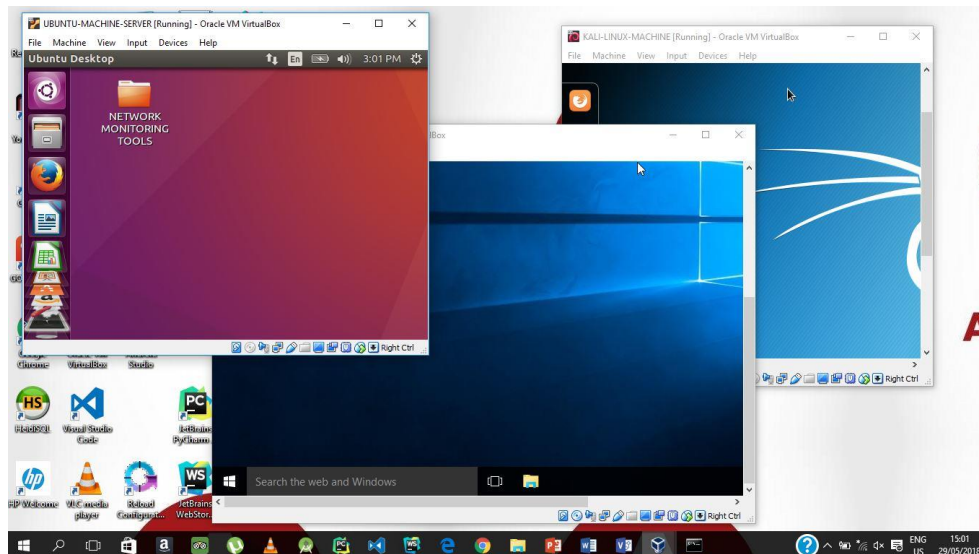


Figure 4.2 – Virtual Machines

Physical Network:

The physical network also included 2 Host Machines (Linux Ubuntu and Windows 10 operating systems) connected to the UGCS department's network platform. Here is a list of the individual hosts within the network, and their corresponding IP Addresses:

- Linux Ubuntu Virtual Machine – 10.1.7.112
- Windows 10 Virtual Machine - 10.1.7.106

4.2.2. Tools Installation and Log file (Datasets) Gathering

At this point, the open-source tools were to be installed within the network (virtual only). Some were successfully installed, others were not. However, for some of those which could not be installed, sample datasets of their log files were gathered through research, and these datasets were used to populate the data stores being monitored by the Sleeper Program (in case of confusion, you may read the operations of the “**Console Application (Sleeper Program) Development**” section for clarity).

Network Monitoring Tools installed so far: Bro, Snort, Nikto

Network Monitoring Tools not installed yet: Ossec, Open VAS, Lynis

Log files retrieved so far:

- BRO (Network Connection Tracking) – conn.log, http.log, ssh.log, ftp.log, smtp.log
- BRO (Network Intrusion Detection) - notice.log, signatures.log
- SNORT (Network Intrusion Detection) – alert.fast, alert.full
- NIKTO (Vulnerability Scanning) – custom_file_name.csv, custom_file_name.xml
- KISMET (Wireless Intrusion Detection) – dhcp.log, dns.log

Log files not retrieved yet:

- OSSEC (Host Intrusion Detection) – alerts.log, alerts.json
- KISMET (Wireless Intrusion Detection) – kismet_xml.log
- LYNIS – (System Auditing) lynis.log

Here, in Fig. 4.3, are some sample log file snippets, where the raw log data formats can be seen; some of the found log file formats include tab-delimited values (tdv), comma-separated values (csv), raw string texts, xml, etc.

```

"STUDENT","197.255.124.12","80","","","Apache/2.4.7 (Ubuntu)"
"STUDENT","197.255.124.12","80","OSVDB-0","GET","/", "Server leaks inodes via ETags, header found with file /, fields: 0x2cf6 0x508937bc3402d "

1331923126.080000 - - - - - HTTP::SQL_Injection_Victim An SQL injection victim was discovered! - 192.168.28.202 - -
1331923345.110000 - - - - - tcp Scan::Address_Scan 192.168.202.108 scanned at least 25 unique hosts on port 443/tcp in 0m1s

1331901000.120000 CyOt6C4vWuJE2n5pDb 192.168.202.79 50483 192.168.229.251 80 tcp http 0.010000 165 214 SF - 0 ShAdfFa 4 381 3 38
1331901000.120000 CRFYpddc0FdD6Zs1 192.168.202.79 46137 192.168.229.254 443 tcp ssl 0.020000 544 1060 SF - 0 ShAdadfr 8 968 13

12/10-18:41:49.864621 [**] (1:2925:3) INFO web bug 0x0 gif attempt [**] (Classification: Misc activity) (Priority: 3) (TCP) 69.194.244.11:80 -> 192.168.76.10:1159
12/10-18:41:49.962983 [**] (1:100000122:1) COMMONITY WEB-MISC mod_jrun overflow attempt [**] (Classification: Web Application Attack) (Priority: 1) (TCP) 192.168.76.10:1156 -> 68.6

```

Figure 4.3 – Sample Log File Formats

4.2.3. Web Server, Client and Console Applications Installation

At this point, network monitoring system was integrated with the setup network, which already had the open-source network monitoring tools installed within it (some of its hosts). 3 main steps were taken in this stage, and they are:

- Setting up the Console (Sleeper Program) Application on the same host(s) within which the monitoring tools were also installed. This step involved installing of the console application and configuring it by setting the file paths or data store locations of the respective monitoring tools installed within each host.
- Setting up of the Web Server Application on the same host(s) within which the sleeper program was setup, so all the sleeper programs could be easily spawned by their corresponding Web Server “supervisors”
- Lastly, setting up of the client applications: Web (Dashboard) Application being hosted by each Web Server setup, and the Mobile Application deployed and installed on mobile devices (Android Platform for now), to be able to run the applications.

After these activities, the integrated network monitoring system was finally ready to undergo Quality Assurance Testing, which will be discussed in the next section.

4.3. System Testing and Results

System Testing, according to the ANSI/IEEE 1059 Standard, is a process of analysing a system or software item in order to detect differences between existing and required conditions such as defects or errors or bugs, and to evaluate the system's features of the system or software item [19]. A special form of System Testing can be called Quality Assurance (QA) Testing, which is that which is performed by a professional engineering team on a given system to ensure that it meets all its requirements and specifications [20]. This chapter is meant to describe major parts of the QA Testing process that the integrated network monitoring system underwent, in order to ensure that it met all its requirements and specifications.

The testing process involved 6 steps for Quality Assurance, and these steps are as follows:

1. Unit Testing – Functional testing on each sub-system
2. Integration Testing – Functional testing on entire system
3. Automated Testing - Continuous Delivery and Integration Setup with “GitLab Auto-DevOps”
4. Acceptance (System) Testing
5. Performance and Stress (Non-Functional) Testing
6. Penetration (Security) Testing

4.3.1. Unit Testing – Functional testing on each sub-system

On this first stage, each sub-component was tested singly, ensuring that all functionalities and features were working without error.

1. Firstly, the MongoDB Database was tested to ensure that new data domain objects could be Created, Updated, Read, and Deleted (CRUD).
2. Web Server was tested using a REST API testing tool called **Postman API**, where each route provided by the Web Server was tested through sample http requests to ensure that the desired http web responses were being returned.
3. The Web Application was also used to test all the functionalities of the Web Server to ensure successful responses on its Graphical User Interface (GUI).
4. The Mobile Application was also used to test all the functionalities of the Web Server to ensure successful responses on its Graphical User Interface (GUI).

- Save Settings success results

The screenshot shows a REST client interface with a PUT request to `http://localhost:8080/api/settings?access_token={{access_token}}`. The request body is a large JSON object. The response status is 200 OK, and the response body is a JSON object indicating success.

```

1 {
2   "success": true,
3   "message": "Settings saved successfully"
4 }

```

- Trigger audit/operation success results

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/api/autosecurity/autoauditing/autoaudits/trigger?access_token={{access_token}}`. The request body is a JSON object. The response status is 200 OK, and the response body is a JSON object indicating success.

```

1 {
2   "success": true,
3   "message": "Auto-Audit 'Network Connection Tracking' has been triggered on all sources successfully"
4 }

```

- Manually send alert message success results

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/api/users/message?access_token={{access_token}}`. The request body is a JSON object. The response status is 200 OK, and the response body is a JSON object indicating success.

```

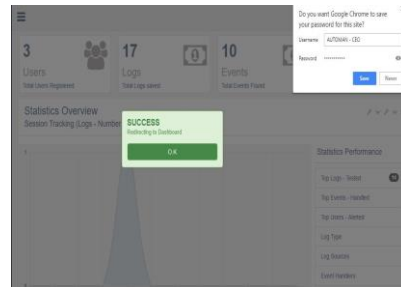
1 {
2   "success": true,
3   "message": "Message Sent by Notification, Email"
4 }

```


4.3.1.2. Web Application Unit Tests

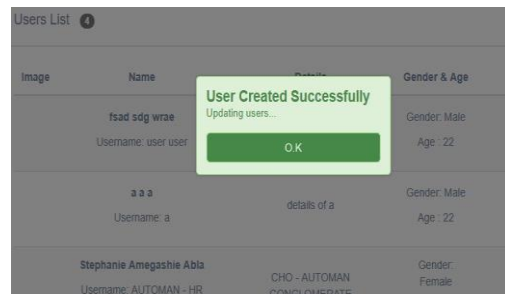
- Successful Login results

The screenshot shows a 'Login Form' with a username field containing 'AUTOMAN - CEO' and a password field with masked characters. Below the fields are buttons for 'Log in', 'Forgot Password?', and 'Register'. At the bottom, there is a 'Dashboard' link and a copyright notice: '©2016 All Rights Reserved. Privacy and Terms'.



- Create New User success results

The screenshot shows a 'Create New User' form with fields for 'Sample Home Address', 'Sample Postal Address', 'AUTOMAN - CEO', 'Password', 'Confirm Password', and 'Main Contact Method'. A 'Submit User Details' button is at the bottom.

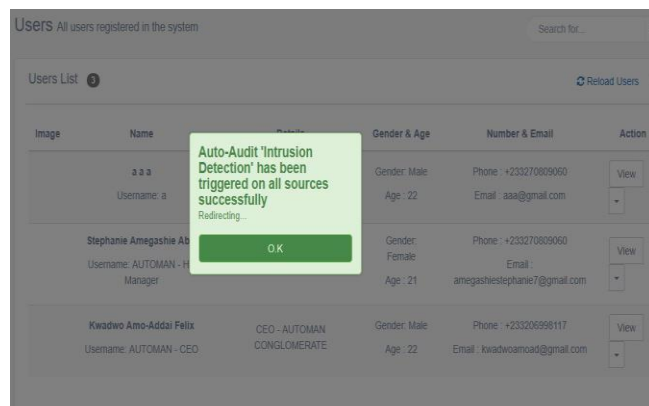


- Save Settings success results

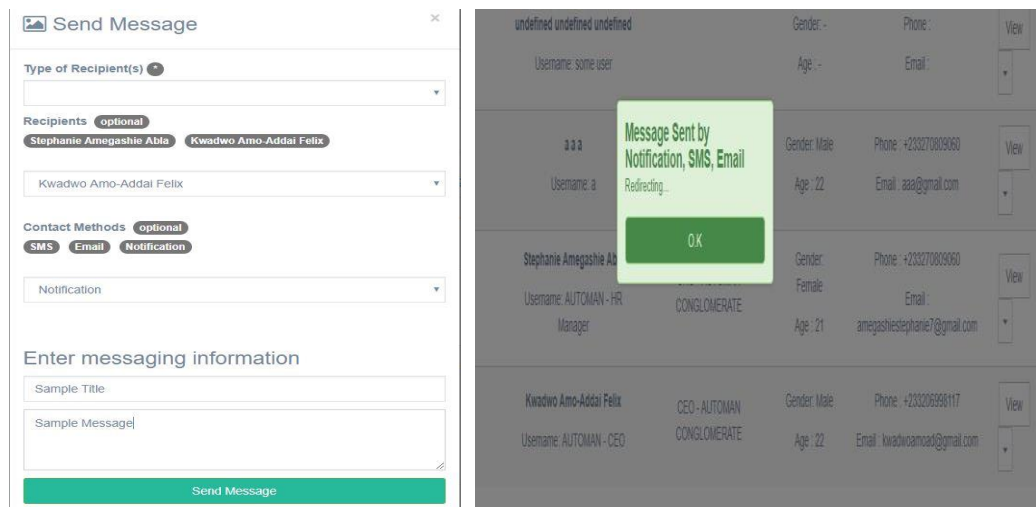
The screenshot shows a 'Security Settings Saved Successfully' message. Below the message are buttons for 'Save Settings' and 'Set Default Settings'.

- Trigger audit/operation success results

The screenshot shows a 'Trigger Auto-Audit' form with fields for 'Internal Auto-Audits', 'External Auto-Audits', 'Source(s) / Tool(s)', and 'Enter Extra Options'. A 'Trigger' button is at the bottom.

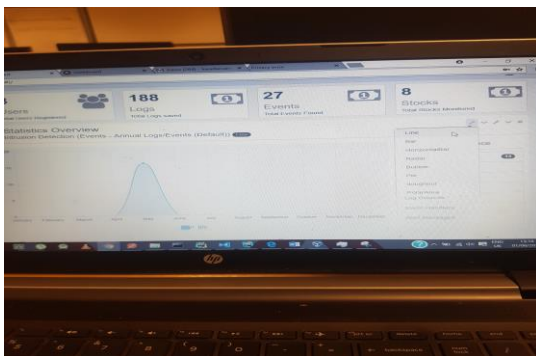


- Manually send alert message success results

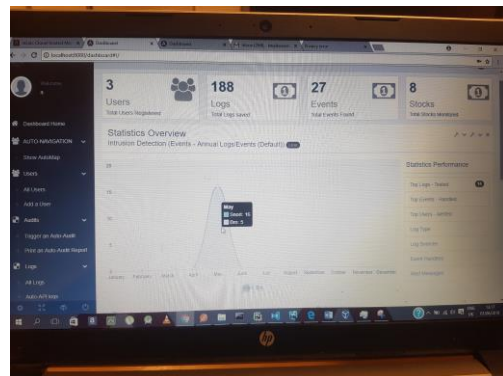


- Show Chart Overview of Network Monitoring Operations

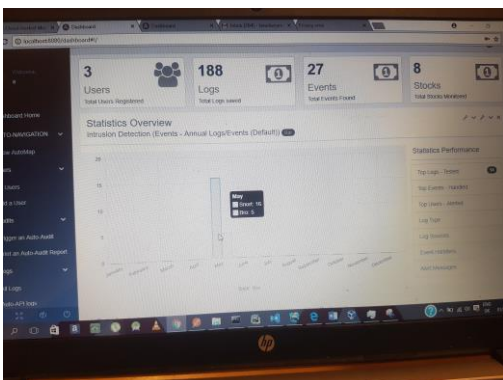
Line Chart (with Chart Options)



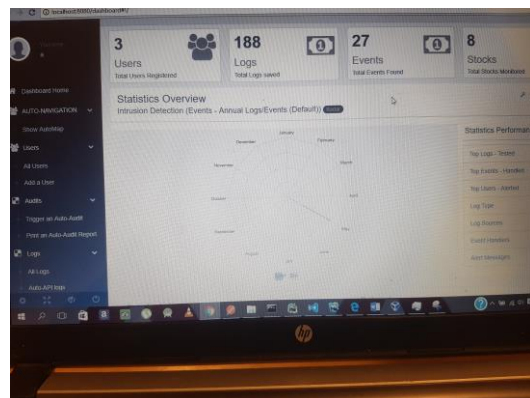
Line Chart



Bar Chart



Radar Chart

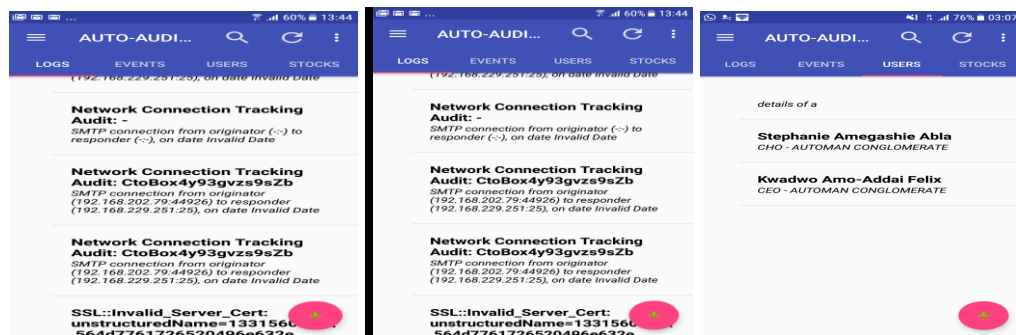


NB: The Chart Overview feature allows for multiple options of viewing charts, and each charting option can be used to view all network monitoring operations, in terms of the number of logs and events detected during such operations throughout the year.

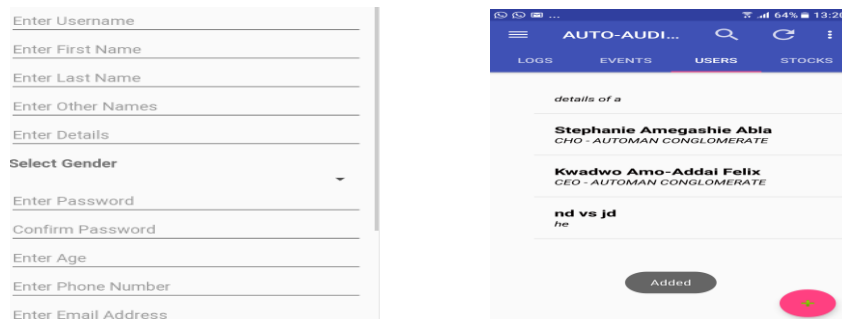
The multiple Chart viewing options are: Line, Bar, Horizontal Bar, Radar, Bubble, Pie, Doughnut, Polar Area Chart Options. Above are some diagrams depicting some of the options.

4.3.1.3. Mobile Application Unit Tests

- Viewing Log/Event and User data



- Create User success results



4.3.2. Integration Testing – Functional testing on entire system

This is the stage where all sub-components (units) are integrated and made to work together to perform a sample audit/operation for testing purposes. A classic example will be the test made in the previous chapter during the “**System Integration**” section, where the sample audit/operation “**Session Tracking**” was performed. The results of the testing can be viewed within that chapter.

4.3.3. Automated Testing - Continuous Delivery, Integration, and Deployment Setup with “GitLab Auto-DevOps”

DevOps, as stated earlier in the previous chapter, is a subset of Cloud Web Services that mainly focus on service platforms used to integrate the “Software Development” and “Operations” (Computer Systems Engineering) sides of a Software Engineering project, by aiding in operations such automated and continuous compilation, delivery, testing, integration, and deployment of any software undergoing the development process. **GitLab Auto-DevOps**, a recently included feature within the GitLab Code Management Version Control Service Platform, is a DevOps feature that completely automates the **CD/CI** (continuous delivery and integration), and testing of a software, and hence seemed to be a very plausible choice for automated testing of this integrated network monitoring system. Here is a clear description of the entire CD/CI process:

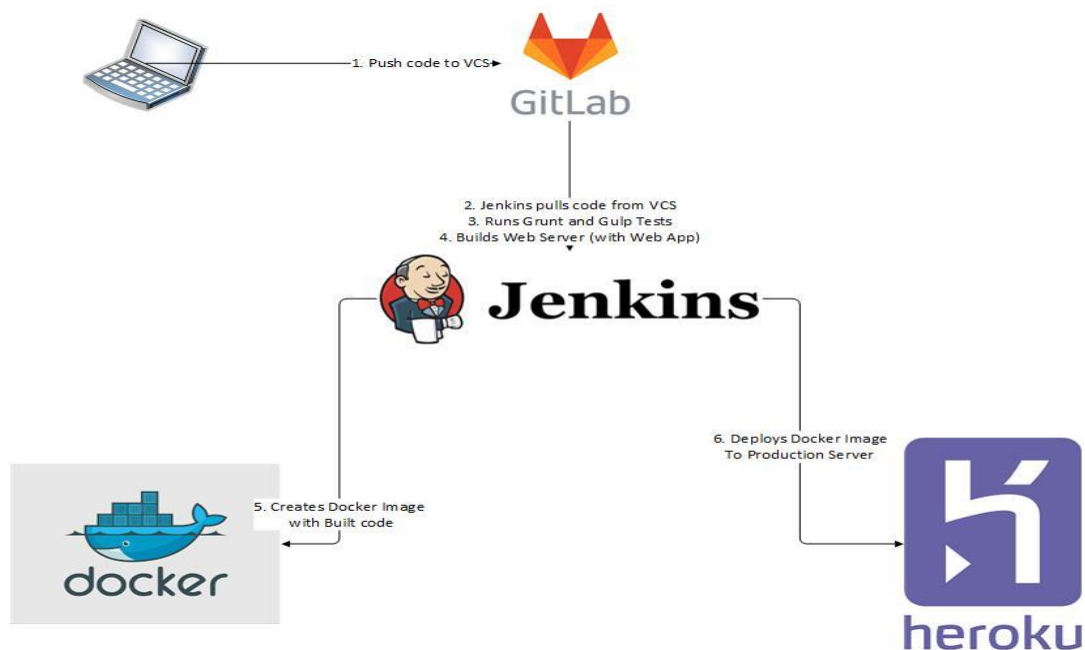


Figure 4.4 – GitLab Auto-DevOps CD/CI Flow

1. First, the code is pushed to the **GitLab VCS** platform, using **Git CLI** (a command-line tool for handling your code base within a VCS repository).
2. **Jenkins**, another DevOps platform, when triggered by a webhook between it and GitLab VCS, pulls the code from the VCS.

3. Jenkins now runs **Grunt** tasks to test the Web Server's services and functionalities using some written test scripts, which send requests to its REST API Web Service routes in order to ensure successful responses.
4. Then after, Jenkins also runs more **Grunt** tasks to build the Web Server, and also some **Gulp** tasks to build the Web (Dashboard) Application. During building of these 2 components, Jenkins minifies (removes unnecessary sections of the code) the entire codebase, making it a lot smaller in size, removes unnecessary files within the project directories, and other functions.
5. Now, a webhook linked between Jenkins and **Docker**, allows Jenkins to enable Docker create a **Docker Image** with the built code. Docker is another DevOps platform that provides the services of packaging Web Server and Application scripts within an image file, called a "Docker Image", which is compatible with and can run on so many Operating Systems (Windows, Linux, MacOS, etc). This helps to greatly reduce dependency management when installing the scripts onto the OS of the host machine.
6. Jenkins now automatically deploys the built Docker Image onto a cloud hosting service called **Heroku**, where the Web Server with the Web Application is automatically kick-started, and can now be easily accessed through the provided url.

NB: Heroku is being used for hosting one of the Web Servers because of its free hosting service package. Therefore, it might not be used when setting up the integrated system within the University of Ghana's network, as Server host machines will already be provided to house the Web Server scripts.

4.3.4. Acceptance (System) Testing

Within this stage, a sample integrated monitoring flow was specified within the configuration settings, using the monitoring tools/sources which had been completely installed at the time, and the monitoring flow is as follows:

- **Vulnerability Scanning**, with **Nikto**, and when it detects the event **Vulnerability Detected**, the integrated system triggers ..
- **Intrusion Detection**, with **Bro** and **Snort**, and when it detects the event **Intrusion Detected**, the integrated system triggers ..
- **Network Connection Tracking**, with **Bro**

These upcoming images depict how the log/event information looks like from the log file stage, as the log file (data store) is being populated by the open-source tool performing the network monitoring operation, to the next stage, when the information has been successfully pre-processed by a Sleeper Program to the standard (JSON) data format, then finally, to the last stage, after it has finally been pre-processed by a Web Server to a more human-readable format and displayed on the client application.

Here are some snapshots of the Web Application results of acceptance testing:

- **Logs List on the Dashboard**

Logs List 188 Reload Logs

Details	Date Created	Action
SSH connection from originator (192.168.202.79:45387) to res	Wed, 30th May '18	View
SSH connection from originator (192.168.202.79:44850) to res	Wed, 30th May '18	View
SSH connection from originator (192.168.202.79:33525) to res	Wed, 30th May '18	View

- **Events List on the Dashboard**

Events List 27 Reload Events

Details	Date Created	Emergency Level	Action
STUDENT (197.255.124.12:80): Apache default file found.	Wed, 30th May '18	default	View
STUDENT (197.255.124.12:80): The X-XSS-Protection header is not define	Wed, 30th May '18	low	View
Scan:Address_Scan: 192.168.202.76 scanned at least 25 unique hosts on	Wed, 30th May '18	default	View
STUDENT (197.255.124.12:80): The anti-clickjacking X-Frame-Options hea	Wed, 30th May '18	low	View

- **Log processing stages (log file -> JSON data -> human-readable format)**

NIKTO (VULNERABILITY SCANNING)

```
STUDENT:197.255.124.12:80:2018-05-30T15:33:04:Apache/2.4.7 (Ubuntu)
STUDENT:197.255.124.12:80:2018-05-30T15:33:04:Server leaks inodes via ETags, header found with file /, fields: 0x2cf6 0x508937bc3402d
```

Name: Vulnerability Detected on host: STUDENT (197.255.124.12:80) on date: Sun Jan 18 1970 15:33:04 GMT+0000 (Greenwich Standard Time)

Details: STUDENT (197.255.124.12:80): Server leaks inodes via ETags, header found with file /, fields: 0x2cf6 0x508937bc3402d

Audit: Vulnerability Scanning

Type: data

Source Type: External

Source: Nikto

Date Created: Thu, 26th April '18

Event: Vulnerability Detected

Emergency Level: default

```
{
  "data": {
    "host": {
      "host_name": "",
      "ip_addr": "",
      "port": ""
    },
    "description": "",
    "timestamp": 0,
    "sources": {
      "Nikto": {
        "osvdbid": "",
        "method": "",
        "uri": ""
      }
    }
  }
}
```

BRO (INTRUSION DETECTION)

```
2018-05-30T15:33:04:STUDENT:197.255.124.12:80:2018-05-30T15:33:04:Apache/2.4.7 (Ubuntu)
2018-05-30T15:33:04:STUDENT:197.255.124.12:80:2018-05-30T15:33:04:Server leaks inodes via ETags, header found with file /, fields: 0x2cf6 0x508937bc3402d
```

Name: Scan:Port_Scan: remote

Details: Scan:Port_Scan: 2001.dbb.c18.202.20c:29ff.fe93.571e scanned at least 15 unique ports of host 2001.dbb.c18.655.20c:29ff.fe1f.96f2 in 0m0s

Audit: Intrusion Detection

Type: data

Source Type: External

Source: Bro

Date Created: Thu, 26th April '18

Event: Intrusion Detected

Emergency Level: default

```
{
  "data": {
    "connection": {
      "id": "",
      "orig_endpoint": {
        "address": "",
        "port": 0
      },
      "resp_endpoint": {
        "address": "",
        "port": 0
      }
    },
    "ids": {
      "protocol": "",
      "timestamp": 0,
      "sources": {
        "Bro": {
          "osvdbid": "",
          "method": "",
          "uri": ""
        }
      }
    }
  }
}
```



```

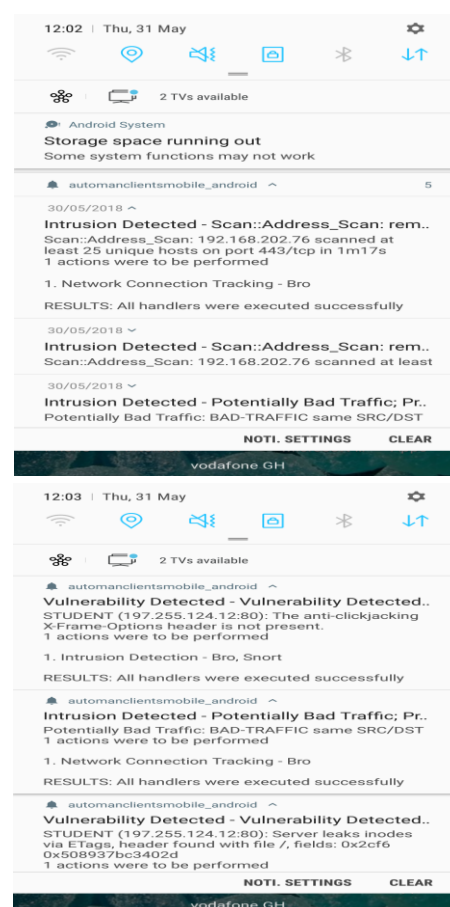
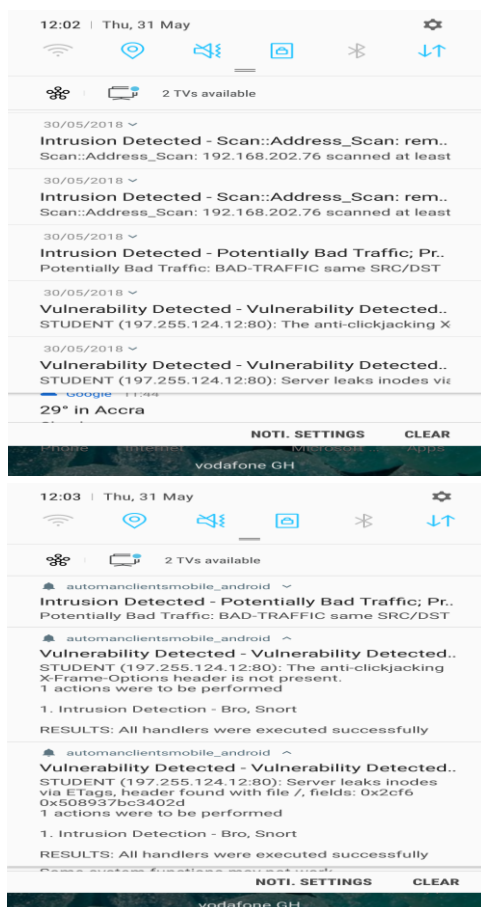
{"id": "69-86902", "info": [{"CVEID": "CVE-2017-1653", "VULN": "web bug hit gdi escape {}"}], [{"Classification": "Risk analysis"}, {"Priority": 3}], [{"CPE": "cpe:/o:redhat:linux:7.0:el7::x86_64"}, {"CPE": "cpe:/o:redhat:linux:7.0:el7::x86_64"}, {"CPE": "cpe:/o:redhat:linux:7.0:el7::x86_64"}], [{"CVEID": "CVE-2017-1653", "VULN": "web bug hit gdi escape {}"}], [{"Classification": "Web Application Attack"}, {"Priority": 3}], [{"CPE": "cpe:/o:redhat:linux:7.0:el7::x86_64"}, {"CPE": "cpe:/o:redhat:linux:7.0:el7::x86_64"}, {"CPE": "cpe:/o:redhat:linux:7.0:el7::x86_64"}]}

```

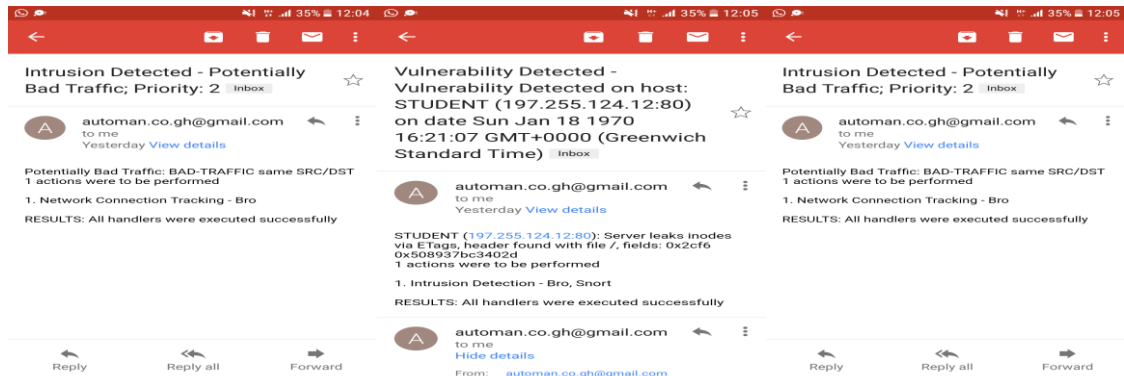
The screenshot displays a Wireshark packet capture of an HTTP GET request. The packet is identified as 'HTTP GET' and '192.168.202.79 80'. The packet details pane shows the 'Network Connection Tracking Audit' section with the following information:

- Name: Network Connection Tracking Audit
- CwQnUo4bRUUd8TaOU6
- Details: HTTP connection from originator (192.168.202.79:50495) to responder (192.168.229.251:80) on date Fri Jan 16 1970 09:58:21 GMT+0000 (Greenwich Standard Time)
- Audit: Network Connection Tracking
- Type: data
- Source Type: External
- Source: Bro
- Date Created: Thu, 26th April '18

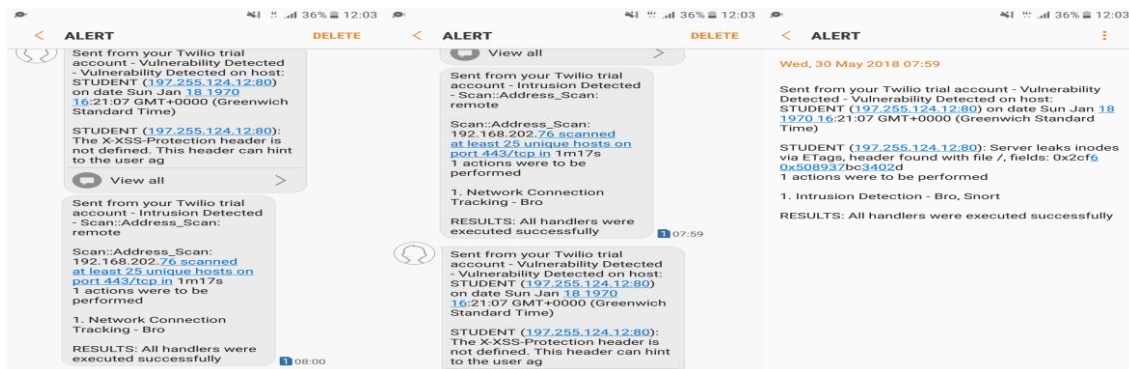
- **Mobile App Notification Alerts**



- **Email Alerts**



- **SMS Alerts**



4.3.5. Performance and Stress (Non-Functional) Testing

At this stage, all functional requirements and specifications of the integrated system have been accounted for, but however, there were some non-functional requirements which the system is supposed to meet, and some of such requirements were robustness and high-performance, even within stressful conditions, good security, flexibility with user-friendliness, etc. In this section, the performance of the integrated system during stressful conditions will be discussed.

As stated earlier, in Chapter 3, during the design of the system's architecture, a component called the **Load Balancer** was introduced. What the Load Balancer does is to distribute the traffic load approaching the Web Servers from the Client applications and Sleeper Programs, and also scale up and out (or scale down and in) the Web Server Workers in a balanced fashion, in order to prevent a Web Server from undergoing too

much stress to handle, causing it to crash. As such, the Load Balancer proves to be one of the salient material requirements for this integrated network monitoring system. However, due to unlimited resources, it cannot be retrieved to be used within the prototype system; instead 2 Web Servers have been setup (one setup locally within the personal laptop of the Project Leader, and the other setup within Heroku, a cloud hosted server).

Therefore, the 2 web servers which were setup were both stress tested by manually populating the data stores of the network monitoring tools at different intervals in order to manipulate the rates at which logs/events are sent to the Web Servers to test the Web Servers' capabilities of processing the data at the corresponding rates. Here is a table depicting some qualitative analysis of the results:

Table 4.1 – System Performance and Stress Testing Results

INTERVAL (seconds)	LOGS/EVENTS PER SECOND	PERFORMANCE (Qualitative)
20	0.05	Very Good
15	0.0667	Moderate
10	0.1	Very Bad
5	0.2	System Crashed (Froze)

4.3.6. Penetration (Security) Testing

For this final testing section, some tools for penetration testing, using the Kali Linux Operating System were used to perform some “Ethical Hacking” attacking schemes on the virtual network setup in order to see how the integrated network monitoring system will behave towards such attacks. For clarity, Penetration Testing is an authorized simulated attack on a computer system, performed to evaluate the security of the system. The test is performed to identify both weaknesses (also referred to as vulnerabilities), including the potential for unauthorized parties to gain access to the system's features and data as well as strengths, enabling a full risk assessment to be completed. Some of the Penetration Testing tools used within the Kali Linux Operating System include:

- Kali Linux OS tools (nmap, aircrack, metasploit, etc) for Penetration Testing of the integrated network system

- **Nmap** – this is an information gathering tool used for the main purpose of “Footprinting”, which is the act of finding a given server’s IP Address, as well as its open ports, services, with the aim of physically infiltrating the server.
- **Aircrack** – this is a Wireless (Wi-Fi) hacking tool used for defeating the security of a wireless local-area network (back-jack wireless LAN or a Wi-Fi network), which have inherent security weaknesses from which wired networks are exempt.
- **Metasploit** – this is a tool for developing and executing exploit code against a remote target machine.
- **Others** – some other tools “played around with” were SEToolkit, Hydra, etc.

4.4. Discussion and Analysis of Results

Looking at all these testing scenarios, one can see that the integrated network monitoring system can perform all its functional requirements and specifications successfully. However, for some non-functional requirements such as robustness and security, there seems to be some room for improvement. Each sub-component, like the Web and Mobile applications, or the Web Servers, could successfully perform all their tasks, as defined in the system block diagram (in chapter 3).

4.5. System Performance Evaluation and Limitations

The integrated system could perform all its functional requirements, some of which include:

- System was capable of integrating multiple operations of multiple network monitoring tools within a small network.
- System could also process the log information from the tools (within the data stores or log files) to a more human-readable format when viewed on the client (both web and mobile) applications.
- System could also detect events or anomalies which occur during monitoring, and then perform remedial actions to handle those anomalies in real-time.
- System could auto-generate real-time alert messages containing information on events or anomalies that occur during monitoring, their emergency levels, and the corresponding remedial actions undertaken to handle the anomalies.

- System could provide the capability of specifying custom configuration settings, to enable system administrators re-configure settings to specify different monitoring flows, and other settings, whenever required, ensuring flexibility in usability of the system.
- System could also provide the capability of managing the data stored within the Central Database, in terms of creating, reading, updating, and deleting all possible data domains (users, logs, and events).
- System also allows system administrators to have an overview of the network monitoring operations performed over the year through various types of charts from which the administrators can choose from. Through this functionality, they can have an idea of the total number of logs and events that occurred during any monitoring operation, and can therefore diagnose the specific parts of the network which seem to be failing the most.

However, the system seemed to be lacking in some scenarios. Here are the limitations of the integrated system that were observed within the testing results:

- System's performance kept degrading as the rate at which the log data being sent to the Web Servers increased (during stress testing). This is because, there were not enough Web Servers to process all the traffic data being sent.
- System could detect almost but not all advanced hacking techniques employed during the penetration testing. Therefore, there seemed to more room for improvement in its security.

CHAPTER 5 – CONCLUSIONS AND RECOMMENDATIONS

5.1. Introduction

In this last chapter, all the conclusions drawn from analysing the results obtained within chapter 4 will be clearly stated. Also, recommendations for expanding and enhancing the integrated system will be discussed, and finally, some general observations made from running the entire project will be stated.

5.2. Conclusions

In conclusion, the Integrated Network Monitoring System was capable of performing all its functional requirements successfully, but not all non-functional requirements. All in all, the system proved to substantially reduce the workload of monitoring any network, whether simple or complex, and can therefore be used by many organizations such as networking departments, telecom companies, etc. Let us look at the specific conclusions about the integrated system as compared to the initial project objectives:

- System was capable of integrating multiple network monitoring operations using the various network monitoring tools within a network, provided it contained a sufficient number of Web Servers for processing.
- System could also process the log information from the tools (within the data stores or log files) to a more human-readable format when viewed on the client (both web and mobile) applications.
- System could also detect events or anomalies which occur during monitoring, and then perform remedial actions to handle those anomalies in real-time.
- System could auto-generate real-time alert messages containing information on events or anomalies that occur during monitoring, their emergency levels, and the corresponding remedial actions undertaken to handle the anomalies.

5.3. Recommendations

Here are also some recommendations for taking care of the system's inabilities and limitations:

- System can be expanded to monitor larger and more complex networks by gathering more network monitoring tools, with their corresponding data stores and log files for continuous integration of their operations.

- System can also be expanded by scaling up and out worker servers, using a Load Balancer to handle log data within a huge network which might be providing very huge traffic data.
- Enhancing the user-friendliness of the client (Mobile and Web) applications for end-users (UGCS system administrators and employees) to enable a more ease-of-use.

5.4. Observations and Challenges

Looking at the latter sections of the previous chapter, the integrated network monitoring system's performance evaluation has been clearly depicted, and it was observed that the system could perform very well in favourable conditions when there was a low rate of traffic load being sent to the Web Servers; but in the case of unfavourable conditions, with a very high rate of incoming traffic, more Web Servers, coupled with a Load Balancer will be required in order to diversify the log data for efficient processing.

However, this section is mainly aimed at allowing the participant(s) involved in the project to express personal observations and challenges experienced throughout the entire project.

As I, Kwadwo Amo-Addai, was the only participant involved in this project (aside my supervisors that is), my personal opinion on this work is that it has pushed me to realise my true potential, skills, and abilities, as well as inculcated discipline within my own personal time management and awareness.

As one can see, this project involves many different topics such as Computer Networking, Cyber-Security, Data Analysis, Artificial Intelligence, Software Architecture and Development, Penetration Testing and Ethical Hacking, etc. Most of these fields I already had extensive knowledge and skills in, due to my personal studies I endeavour to take frequently. However, with some particular fields within Cyber-Security, the network monitoring tools to be specific, I encountered some challenges where I had to research on each of the various tools (like Bro, Snort, Suricata, Kismet, Ossec, Nikto, OpenVAS, Lynis, etc), as well as their network monitoring operations (like Network Connection Tracking, Intrusion Detection, Vulnerability Scanning, System Auditing, etc), in order to get a clear understanding on them to know how exactly I could design, develop, and implement an Integrated Network Monitoring

System or Platform that was fully-generic, so it seamlessly harmonize the tools and their operations, whilst completely decentralizing their data using the Centralized Database in order to successfully cluster the data.

So to cut it all short, I truly believe that any individual can dare to be innovative enough to achieve any aim and solve any problem, no matter how impossible.

REFERENCES

- [1] Colin Pattinson. A study of the behaviour of the simple network management protocol. In Olivier Festor and Aiko Pras, editors, DSOM, pages 305–314. INRIA, Rocquencourt, France, 2001.
- [2] "What is Syslog? Understanding the Complexities of Network Management". networkmanagementhub.com
- [3] Scarfone, Karen; Mell, Peter (February 2007). "Guide to Intrusion Detection and Prevention Systems (IDPS)" (PDF). Computer Security Resource Center. National Institute of Standards and Technology (800–94). Retrieved 1 January 2010.
- [4] "Penetration Testing vs. Vulnerability Scanning". www.tns.com. Retrieved 2016-12-07
- [5] National Institute of Standards and Technology (September 2008). "Technical Guide to Information Security Testing and Assessment" (PDF). NIST. Retrieved 2017-10-05.
- [6] "Bro: An Open Source Network Intrusion Detection System" Retrieved 3 April 2018 – via GitHub.
- [7] Mohan Krishnamurthy; et al. (2008). "4. Introducing Intrusion Detection and Snort". How to Cheat at Securing Linux. Burlington, MA: Syngress Publishing Inc. Retrieved 2010-06-24.
- [8] Jonkman, Matt (2009-12-31). "Suricata IDS Available for Download!". Seclists.org. Retrieved 2011-11-08.
- [9] Murray, Jason. "An Inexpensive Wireless IDS using Kismet and OpenWRT". SANS
- [10] "OSSEC Architecture". OSSEC Project Team. 2017. Retrieved 2018-05-10. Institute. Retrieved 9 March 2016.
- [11] "Data file distributed with Nikto with non-Open Source licence notice at the top".
- [12] LeMay, Renai (2005-10-06). "Nessus security tool closes its source". CNet.
- [13] "Lynis 2.2.0 Released – Security Auditing and Scanning Tool for Linux Systems". Ravi Saive, tecmint.com. 2016-03-18. Retrieved 2017-03-20.
- [14] Raywood, Dan (April 24, 2015). "HP partner with AlienVault on Cyber Threat-Sharing Initiative". ITPortal.com. Retrieved November 8, 2015.
- [15] The Cacti Group, Inc. Cacti: The complete rrdtool-based graphing solution. <http://www.cacti.net>. Accessed June 6, 2015.

- [16] The Icinga Project. Icinga: Open source monitoring. <http://www.icinga.org>. Accessed June 3, 2015.
- [17] Nagios. The Industry Standard In IT Infrastructure Monitoring. <http://www.nagios.org>. Accessed June 3, 2015.
- [18] Information Sciences Institute, University of Southern California. Internet protocol. RFC 791, RFC Editor, <http://www.ietf.org/rfc/rfc791.txt>, September 1981.
- [19] Ambysoft: The Agile System Development Life Cycle (SDLC), <http://www.ambysoft.com/essays/agileLifecycle.html>. Accessed June 6 2015.
- [20] Stephen Few. What is a dashboard? In Information Dashboard Design: The Effective Visual Communication of Data, page 34. O'Reilly Media, Inc, 2006.
- [21] Engin Kirda; Somesh Jha; Davide Balzarotti (2009). Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009, Saint-Malo, France, September 23–25, 2009, Proceedings. Springer. p. 162. ISBN 978-3-642-04341-3. Retrieved 29 June 2010.
- [22] "Vulnerability Assessment" (PDF). www.scitechconnect.elsevier.com. Retrieved 2016-12-07
- [23] Ribeiro, Anna (9 February 2017). "AlienVault announces USM Anywhere unified security management platform; achieves AWS Advanced Technology Partner st". Computer Technology Review. Retrieved 7 October 2017.
- [24] Goldman, George (17 January 2007). "The network graphing solution Cacti was designed to provide more ease of use than RRDtool and more flexibility than MRTG". ISP-Planet. Retrieved 16 March 2012.
- [25] Mobily, Tony (27 April 2012). "Nagios Vs. Icinga: the real story of one of the most heated forks in free software". Free Software Magazine.
- [26] "NetSaint Change Log". 2002-03-01. Archived from the original on 2006-05-01.
- [27] Jeffrey Carr (2007-06-05). "Snort: Open Source Network Intrusion Prevention". Retrieved 2010-06-23.
- [28] "New Open Source Intrusion Detector Suricata Released". Slashdot. 2009-12-31. Retrieved 2011-11-08.
- [29] Thejdeep, G. "Detecting Rogue Access Points using Kismet". In Communications and Signal Processing: 0172–0175.
- [30] "About". OSSEC Project Team. 2017. Retrieved 2018-05-10.
- [31] "lynis: Lynis - CONTRIBUTORS - doxygen documentation - Fossies Dox". M. Boelen, fossies.org. 2017-03-15. Retrieved 2017-03-20.


```

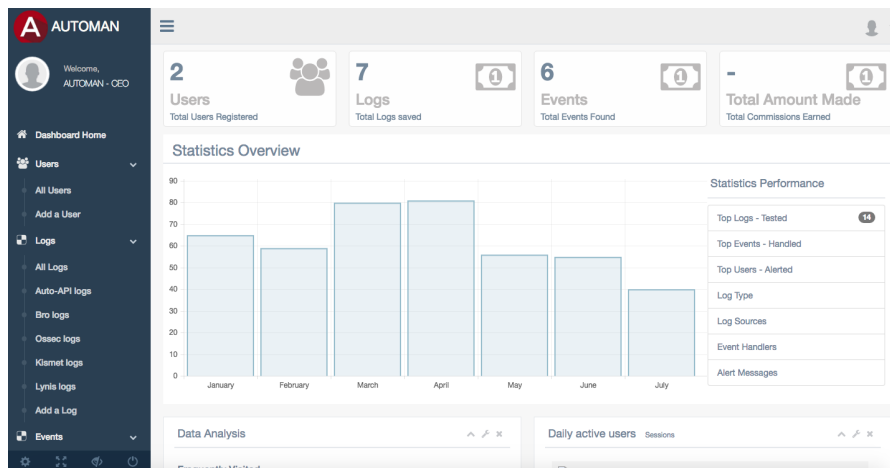
{
  "type": "data",
  "source_type": "Internal",
  "source": "Auto-API",
  "autoevents": [],
  "date_created": "2018-02-23T15:50:54.894Z",
  "id": "5a90385ebcdd4a1350416fff8",
  "name": "Session Tracking audit pertaining to user: Kwadwo Amo-Addai Felix",
  "details": "User (Kwadwo Amo-Addai Felix) has logged in with a device (), on date Fri Feb 23 2018 07:50:54 GMT-0800 (Pacific Standard Time) with access token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3fWFuQ010I1YTkzMzZjZGQ0YTEzNTA0MTZmZjYiLCJpYXQiOiJlMTk0MDEwNTQsImV4cCI6MTUxOTQwMjQ5NH0.9gZaiaKtCws~crHNa1K1eZcdLcuhhwg8557gGvxrn50",
  "autoaudit": "Session Tracking",
  "data": {
    "user": {
      "id": "5a90381fbcdd4a1350416fff6",
      "full_name": "Kwadwo Amo-Addai Felix"
    },
    "access_token": {
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3fWFuQ010I1YTkzMzZjZGQ0YTEzNTA0MTZmZjYiLCJpYXQiOiJlMTk0MDEwNTQsImV4cCI6MTUxOTQwMjQ5NH0.9gZaiaKtCws~crHNa1K1eZcdLcuhhwg8557gGvxrn50",
      "expiresInMinutes": 1440
    },
    "device": {
      "id": ""
    },
    "login": true,
    "logout": false,
    "timestamp": 1519401054871
  },
  "updatedAt": "2018-02-23T15:50:54.904Z",
  "createdAt": "2018-02-23T15:50:54.904Z",
  "_v": 0
}

```

Appendix A2 – Web (Dashboard) Application Images

- Login Page

- Home Page View (with Chart Summary)



- Settings Specification

Internal Auto-Audits

Auto-Audit: Session Tracking

Possible Anomalies/Events:

User Multiple Logins Unrecognized Device

Handler Type:

Real Time

Handler Method

Correlation Analysis

Sources

Auto-API Brs Cmscc

Cmscc

EVENT HANDLER SETTINGS

Auto-Event Handler Actions

Blocklist user's access token Delete user from System's Database

Auto-Event Emergency-Level default Handler Actions

Blocklist user's access token

Auto-Event Emergency-Level low Handler Actions

Auto-Event Emergency-Level medium Handler Actions

Auto-Event Emergency-Level high Handler Actions

Emergency Level Settings (Correlation Rates - Events/Sec)

Number of Events

5

default (Duration - seconds)

0

low (Duration - seconds)

10

medium (Duration - seconds)

5

high (Duration - seconds)

1

Auto-Generate Alert Message

☒

Default Subject

Default Message

Auto-Event Alert Extra Data Properties:

Auto-Event Alert Type of Recipient(s): Users

Contact Methods

Notification Email SMS

Recipients

Random Auto-Add Email

- Trigger Audit/Operation (Manually)

Trigger Auto-Audit

Internal Auto-Audits

External Auto-Audits

Source(s) / Tool(s) optional

Enter Extra Options

Trigger

Close

- Adding/Updating Users

User Form
×

Enter details of the user below

Kwadwo

Amo-Addai

Felix

optional

Gender

Male

22

CEO - AUTOMAN CONGLOME

+233206998117

kwadwoamoad@gmail.com

Sample Home Address

optional

Sample Postal Address

optional

AUTOMAN - CEO

Password

Confirm Password

Main Contact Method

Notification

Submit User Details

Close

- Adding/Updating Logs (left) and Events (right)

Log Form
×

Enter details of the Log below

Sample Log

Sample Log Details

Enter Log Data (Use required Json Data format for the corresponding Audit)

Audit

System Auditing

Log Type

data

Type of Source

External

Source

Lynis

Enter details of the Event below

Sample Event

Sample Event Details

Enter Event Data (Use required Json Data format for the corresponding Audit)

Audit

Intrusion Detection

Log Type

data

Type of Source

External

Source

Bro

Event

Intrusion Detected

Emergency Level

low

- Viewing Users

Users All users registered in the system Search for... Go!

Users List 2 Reload Users

Image	Name	Details	Gender & Age	Number & Email	Action
	Stephanie Amegashie Abia Username: AUTOMAN - HR Manager	CHO - AUTOMAN CONGLOMERATE	Gender: Female Age : 21	Phone : +233270809060 Email : amegashiestephanie7@gmail.com	View
	Kwadwo Amo-Addai Felix Username: AUTOMAN - CEO	CEO - AUTOMAN CONGLOMERATE	Gender: Male Age : 22	Phone : +233206998117 Email : kwadwoamoad@gmail.com	View

- Viewing Logs

Logs All logs registered in the system Search for... Go!

Logs List 7 Reload Logs

Details	Date Created	Action
Details : User (Kwadwo Amo-Addai Felix) has logged in with a device ()	Thu, 8th March '18	View
Details : User (Kwadwo Amo-Addai Felix) has logged in with a device ()	Thu, 8th March '18	View

- Viewing Events

Events All events registered in the system Search for... Go!

Events List 8 Reload Events

Details	Date Created	Emergency Level	Action
User: (Kwadwo Amo-Addai Felix) has had 2 logins with 2 devices	Fri, 9th March '18	default	View
Some Klismet AutoEvent ohhhh	Fri, 9th March '18	default	View

- Sending Messages to Users (Manually)

Send Message
×

Type of Recipient(s)

Users

Recipients optional

Kwadwo Amo-Addai Felix

Stephanie Amegashie Abia

Stephanie Amegashie Abia

Contact Methods optional

Notification

SMS

Email

Email

Enter messaging information

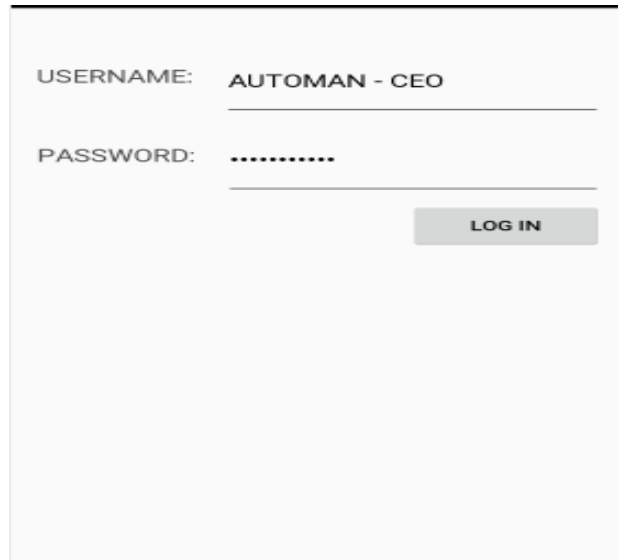
Some title / subject

Some message to send ...

Send Message

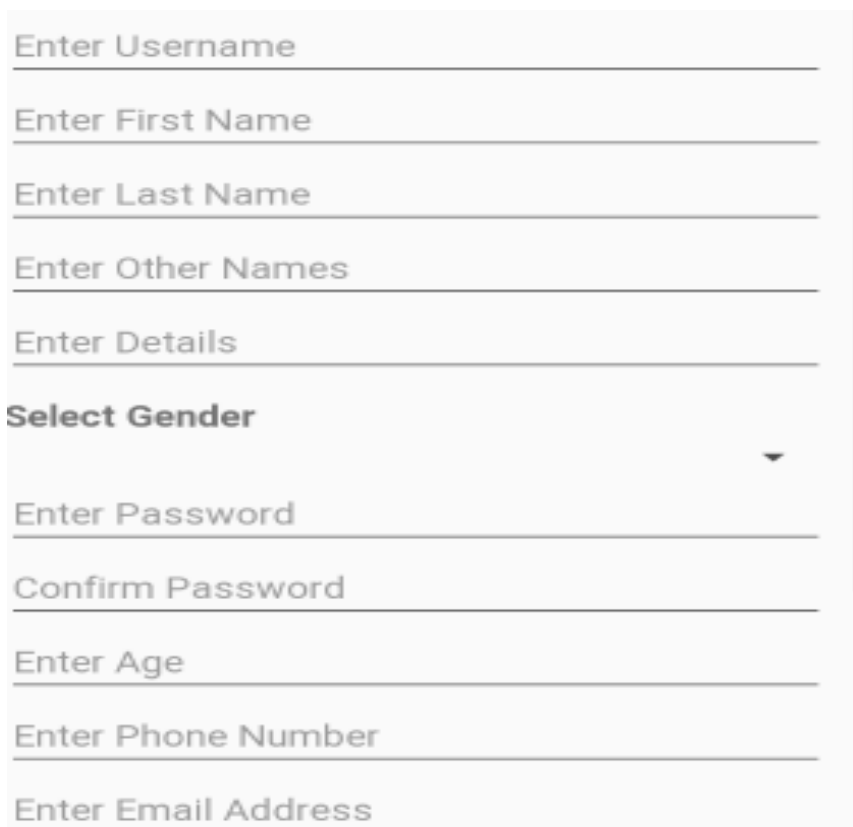
Appendix A3 - Mobile Application Images

- Login Page



A screenshot of a mobile application login page. It features a light gray background with a white rectangular form. Inside the form, there are two input fields. The first is labeled 'USERNAME:' and contains the text 'AUTOMAN - CEO'. The second is labeled 'PASSWORD:' and contains a series of dots. To the right of the password field is a gray button with the text 'LOG IN' in white capital letters.

- Add/Update User



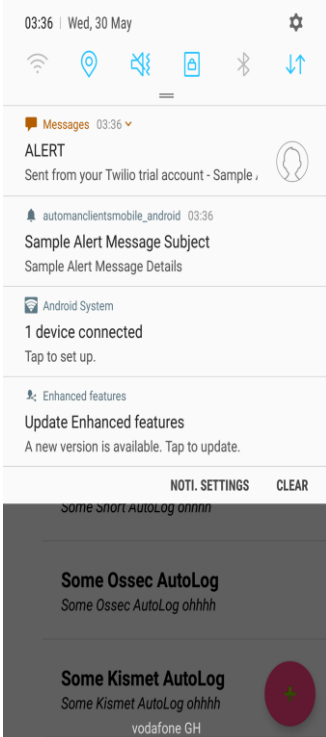
A screenshot of a mobile application form for adding or updating a user. The form has a light gray background and a white rectangular area with several input fields. The fields are labeled as follows: 'Enter Username', 'Enter First Name', 'Enter Last Name', 'Enter Other Names', 'Enter Details', 'Select Gender' (with a dropdown arrow), 'Enter Password', 'Confirm Password', 'Enter Age', 'Enter Phone Number', and 'Enter Email Address'.

- Add/Update Log & Event

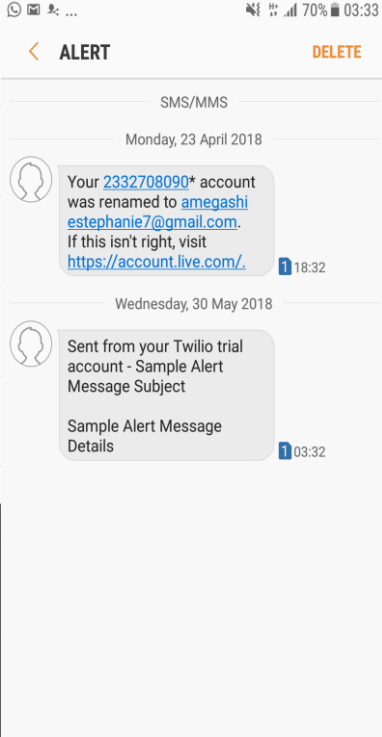
Enter Name	Enter Name
Enter Details	Enter Details
Enter Data (JSON Format)	Enter Data (JSON Format)
Select Type	Select Type
Select Source Type	Select Source Type
Select Source	Select Source
Select Audit	Select Audit
Select Event(s)	Select Event
CANCEL	Select Emergency Level
SAVE	

- Receiving Alert Messages

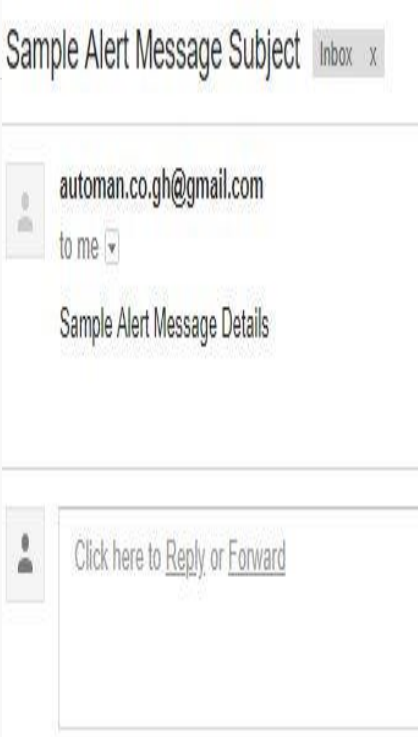
Notification



SMS



Email



APPENDIX B – CODE SNIPPETS

Appendix B1 – Web Server Application Code Snippet

```
// Set default node environment to development
process.env.NODE_ENV = process.env.NODE_ENV || 'development';

var express = require('express');
var mongoose = require('mongoose');
var pid = process.pid;

var config = require('./server/config/environment');

// Connect to database
mongoose.connect(config.mongo.uri, config.mongo.options)
.then(() => {
  console.log("Connected to Database server (" + config.mongo.uri + ") successfully...");
})
.catch((err) => {
  console.log("Could not connect to Database server (" + config.mongo.uri + ") successfully...");
  if(err) console.log("ERROR -> " + JSON.stringify(err));
});

// Populate DB with sample data
if (config.seedDB) require('./server/config/seed');
// Test DB with sample data
if (config.testDB) require('./server/config/unit.testing');

var app = express();
var path = require('path');

app.use('/assets', express.static(path.join(__dirname + '/dashboard')));
// app.use('/assets', express.static(path.join(__dirname + '/dash')));

//CORS middleware
```



```

var allowCrossDomain = function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,PATCH');
  res.header('Access-Control-Allow-Headers', 'Content-Type');
  next();
};
app.use(allowCrossDomain);

app.get('/dashboard', function(req, res) {
  res.sendFile(path.join(__dirname + '/dashboard/dashboard.html'));
  // res.sendFile(path.join(__dirname + '/dash/dash.html'));
});

// CHANGE THIS TO https WHEN GOING LIVE IN PRODUCTION
// ENVIRONMENT
var server = require('http').createServer(app);
var socketio = require('socket.io')(server, {
  serveClient: (config.env === 'production') ? false : true,
  path: '/socket.io-client'
});

require('./server/config/express')(app);
require('./server/routes')(app);
require('./server/config/socketio')(socketio);
require('./server/config/server.scaling')(app);

// Start server - https://localhost:8080 (development/test) &
// https://automanghana.herokuapp.com (production)
// server.listen(config.port, config.ip, function () {
//   console.log('Server (%s) listening on port %d, in %s mode with process %d',
// config.ip, config.port, app.get('env'), pid);
// });
server.listen(config.port, function () {
  console.log('Server (%s) listening on port %d, in %s mode with process %d', config.ip,
config.port, app.get('env'), pid);
}); // THIS IS FOR WINDOWS (FOR MAC, JUST RUN -> mongo)

```

```
// "C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe" - RUN MONGODB
INSTANCE
// "C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe" - RUN MONGODB SHELL
/ CLI
```

```
// Expose app
module.exports = app;
```

Appendix B2 – Web (Dashboard) Application Code Snippet

```
angular.module('dashboard', [
    'ui.router',
    'ui.bootstrap',
    'ngStorage',
    'ui.select',
    'angular-growl',
    'angularMoment',
    'angular-loading-bar',
    'angular-cache',
    'mgo-angular-wizard',
    'chart.js',
    'infinite-scroll',
    'ngFileUpload',
    'templates'
])
.config(['$stateProvider', '$urlRouterProvider', '$httpProvider',
'CacheFactoryProvider',
function ($stateProvider, $urlRouterProvider, $httpProvider,
CacheFactoryProvider) {

    console.log("APP IS SETTING UP!!!");

    $urlRouterProvider.when('/', '/login');
    $urlRouterProvider.otherwise('/login');
```

```

//Http Interceptor to check auth failures for xhr requests
$httpProvider.interceptors.push('HttpInterceptor');

angular.extend(CacheFactoryProvider.defaults, {
    maxAge: 24 * 60 * 60 * 1000,
    storageMode: 'localStorage',
    recycleFreq: 12 * 60 * 60 * 1000,
    deleteOnExpire: 'aggressive',
    storagePrefix: 'mg_'
});

});

angular.module('dashboard').run(['$rootScope', '$state', '$stateParams', '$localStorage',
'Constants', 'SettingService',
function ($rootScope, $state, $stateParams, $localStorage, Constants, SettingService)
{

    console.log("APP HAS STARTED!!!");

    $rootScope.$state = $state;
    $rootScope.$stateParams = $stateParams;
    $rootScope.image_base_url = Constants.image_base_url;

    if (angular.isDefined($localStorage['access_token'])) {
        $rootScope.access_token = $localStorage['access_token'];
    }
    if (angular.isDefined($localStorage['user'])) {
        $rootScope.user = $localStorage['user'];
    }
    $localStorage['company'] = {};
    $rootScope.company = {};
    // if (angular.isDefined($localStorage['company'])) {
    //     $rootScope.company = $localStorage['company'];
    // }
    if (angular.isDefined($localStorage['companySetting'])) {

```

```

    $rootScope.companySetting = $localStorage['companySetting'];
  }

  $rootScope.$on('$stateChangeStart', function (event, toState, toParams, fromState,
fromParams) {
    $rootScope.pageLoading = true;
  });

  /*Set Sound Setting for Alert JS*/
  AlertJS.setSetting("sound", true);

  var userNotLoggedIn = $('.showBeforeAuthentication'),
      userIsValidated = $('.showAfterAuthentication'),
      body = $('body');

  $rootScope.$on('$stateChangeSuccess', function (event, toState, toParams,
fromState, fromParams) {
    if ($state.is('login') || !$rootScope.isLoggedIn()) {
      console.log("USER NOT LOGGED IN !!!")
      $rootScope.deleteUserAndCompanyData();
      userNotLoggedIn.show();
      body.addClass('login_bg_color');
      userIsValidated.hide();
    } else {
      userNotLoggedIn.hide();
      body.removeClass('login_bg_color');
      userIsValidated.show();
    }
  });

  $rootScope.$on('$viewContentLoaded', function (event) {
    $rootScope.pageLoading = true; // YOU MIGHT HAVE TO REMOVE THIS
    LATER THOUGH
  });

  $rootScope.$on('$viewContentLoaded', function (event) {

```

```

    $rootScope.pageLoading = false;
  });

  $rootScope.getObjectSize = function (obj) {
    return _.size(obj);
  };

  if (!String.prototype.startsWith) {
    String.prototype.startsWith = function (searchString, position) {
      position = position || 0;
      return this.substr(position, searchString.length) === searchString;
    };
  }

  $rootScope.validateDataSecurity = function (type, action, data) {
    var y = "all fields";
    return {success: true, security: y};
  };

  $rootScope.genderOptions = ["Male", "Female"];

  $rootScope.isLoggedIn = function () {
    //
    console.log("ACCESS TOKEN")
    console.log(JSON.stringify($localStorage.access_token))
    console.log(JSON.stringify($rootScope.access_token))
    //
    console.log("USER")
    console.log(JSON.stringify($localStorage.user))
    console.log(JSON.stringify($rootScope.user))
    //
    return (angular.isDefined($localStorage['access_token']) &&
    angular.isDefined($rootScope['access_token']) &&
    angular.isDefined($localStorage['user']) &&
    angular.isDefined($rootScope['user']) &&

```

```

        angular.isDefined($localStorage['companySetting'])
angular.isDefined($rootScope['companySetting']) );
    };

```

```

$rootScope.deleteUserAndCompanyData = function () {
    //
    delete $localStorage.access_token;
    delete $localStorage.user;
    delete $localStorage.company;
    //
    delete $rootScope.access_token;
    delete $rootScope.user;
    delete $rootScope.company;
    //
    delete $localStorage.companySetting;
    delete $rootScope.companySetting;
};

```

```

$rootScope.getCompanySetting = function (cb) {
    if (angular.isDefined($rootScope.companySetting)
angular.isDefined($localStorage.companySetting)) {
        console.log("COMPANY SETTINGS HAVE ALREADY BEEN DEFINED
...")
        cb(true);
    } else {
        console.log("GETTING COMPANY SETTINGS ...")
        SettingService.getCompanySetting(cb);
    }
};

```

```

$rootScope.getDashboardArray = function (x) {
    switch (x) {
        case "autoauditSources":
            return [
                "Auto-API",

```

```

        "Bro",
        "Snort",
        "Ossec",
        "Kismet",
        "Lynis",
        "Nikto"
    ];
    case "autoauditOptions":
        return [
            "Session Tracking",
            "Stock Market Monitoring",
            "Geo-Spatial Monitoring",
            "Network Connection Tracking",
            "Intrusion Detection",
            "Vulnerability Scanning",
            "System Auditing",
            "AntiVirus Scanning"
        ];
    case "":
        break;
    default:
        var res = SettingService.getDashboardArray(x);
        return res || [];
    }
};

$rootScope.getDashboardValue = function (x) {
    switch (x) {
        case "":
            break;
        default:
            // console.log("GETTING DASHBOARD ARRAY -> " + x);
            var res = SettingService.getDashboardValue(x);
            // console.log("RESPONSE -> " + JSON.stringify(res));
            return res || { };
    }
}

```

```

    };

    $rootScope.getAutoAuditAutoEventOptions = function (autoaudit) {
        if (angular.isDefined($rootScope.companySetting)) {
            var autoaudits =
$rootScope.companySetting.AUTO_AUDITINGSettings.autoaudits;
            for (var x of ["Internal", "External"]) {
                if (autoaudits[x].hasOwnProperty(autoaudit)) {
                    return autoaudits[x][autoaudit].autoeventOptions;
                }
            }
        }
        console.log("SETTINGS NOT DEFINED SO CANNOT GET AUTO-EVENT
OPTIONS ...")
        return [];
    };

    $rootScope.reloadAllData = function (cb) {
        // $rootScope.reloadCompanySetting(); // FIND A WAY TO RELOAD ALL
        THE DATA BEFORE YOU CALL cb() WHICH'LL MOST LIKELY MOVE STATE
        INTO HOME PAGE
        // $rootScope.reloadAutoLogs(); $rootScope.reloadAutoEvents();
        $rootScope.reloadUsers(); $rootScope.reloadStocks();
        console.log("RELOADING ALL DATA, BUT THIS MUST BE
IMPLEMENTED TO WORK PERFECTLY THOUGH ...")
        cb();
    };

    console.log("IS LOGGED IN -> " + $rootScope.isLoggedIn());

    });

```


Appendix B3 – Mobile Application Code Snippet

```
private static Activity currentActivity = null;

public Activity getCurrentActivity() {
    Log.e("CURRENT ACTIVITY ->", (currentActivity != null) ?
currentActivity.toString() : " IS NULL, SORRY :(");
    return currentActivity;
}

private static AutomanApp app = new AutomanApp();

public static AutomanApp getApp() {
    return app;
}

@Override
public void onCreate() {
    super.onCreate();

    Log.e("ONCREATE FUNCTION", "THE APP HAS STARTED ...");

    this.setupPrefs();
    this.setupServer();
    this.setupNotificationHandler();
    this.setupSettingsHandler();
    this.setupCacheData();

    registerActivityLifecycleCallbacks(this);
}
```

Appendix B4 – Console (CLI) Application (Sleepers Program) Code Snippet

```
class AutoAuditingSleepersApp:

    def __init__(self):
        self._loop = None
        #
        self._isActive = False
        self._api_url = ""
        self._access_token = ""
        self._data = {}
        #
        self._settingsHandler = None
        self._fileSystem = None
        self._autoParser = None
        self._server = None
        #
        self._cli = None

    def validateParameters(self, params): # PERFORM SOME VALIDATIONS ON
params
        print("FINALLY -> " + json.dumps(params))
        if "access_token" in params and "settings" in params:
            x = params["settings"]
            if ("autoaudit" in x or "action" in x) and ("sources" in x):
                return True
        return False

    def setup(self, params):
        try:
            self.cli = CLI()
            params = self.cli.parseOptions(params)
            if self.validateParameters(params):
                if sys.platform == 'win32':
                    print("WINDOWS (Win32) SYSTEM")
                    self.loop = asyncio.ProactorEventLoop()
```

```

        asyncio.set_event_loop(self.loop)
    else:
        self.loop = asyncio.get_event_loop()
    print("PARAMS VALIDATED, NOW SETTING UP")
    self.api_url = params["url"]
    self.access_token = params["access_token"]
    self.settingsHandler = SettingsHandler(params["settings"])
    self.fileSystem = AccessFileSystem(params)
    # self.autoParser = AutoParser()
    # self.server = AccessAutomanAPI(self.api_url, self.access_token)
    #
    self.isActive = True
    return self
except Exception as e:
    print("ERROR OCCURRED DURING SETUP -> " + e.message)
    print(e)
    self.isActive = False
    return None

def start(self):
    if self.isActive:
        self.cli.showOutput("AUTO-AUDITING SLEEPER HAS OFFICIALLY
STARTED!!!")
        settings, autoauditOrActionType, autoauditOrAction =
self.settingsHandler.settings, "", ""
        if "autoaudit" in settings:
            autoauditOrActionType = "autoaudit"
            autoauditOrAction = settings[autoauditOrActionType]
        elif "action" in settings:
            autoauditOrActionType = "action"
            autoauditOrAction = settings[autoauditOrActionType]
        sources = settings["sources"]
        # NOW, CONTACT AccessFileSystem.py TO PERFORM
        AUTOAUDIT/ACTION
        self.cli.showOutput(

```

```

        "Performing " + autoauditOrActionType + " " + autoauditOrAction + " with
sources " + str(
    sources.keys()))
    success = self.loop.run_until_complete (
        self.fileSystem.performAutoAuditOrAction(autoauditOrActionType,
autoauditOrAction, sources)
    )
    # success = self.fileSystem.performAutoAuditOrAction(autoauditOrActionType,
autoauditOrAction, sources)
    self.cli.showOutput("")
    if success:
        self.cli.showOutput(
            autoauditOrActionType + " " + autoauditOrAction + " has been performed
successfully by all sources ...")
    else:
        self.cli.showOutput("Sorry, " + autoauditOrActionType + " " +
autoauditOrAction + " could not be performed successfully")
        self.cli.showOutput("")
    else:
        self.cli.showOutput("AUTO-AUDITING SLEEPER CANNOT START
BECAUSE IT IS NOT ACIVE")

def stop(self):
    if self.isActive: # PERFORM FUNCTIONS TO END THE EXECUTION OF
THIS APP
        self.isActive = False
        print("ABOUT TO END SLEEPER, HALTING FILE MONITORING ...")
        self.fileSystem.stopFileMonitoring()
    else:
        pass

# THIS TRICK HELPS MAKE THIS CLASS A SINGLETON
appInstance = AutoAuditingSleeperApp()

```