2015-11-01

Deriving System Vulnerabilities Using Log Analytics

Matthew Somers Higbee
Brigham Young University

Deriving System Vulnerabilities

Using Log Analytics

Matthew Somers Higbee

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dale C. Rowe, Chair
Derek L. Hansen
Joseph J. Ekstrom

School of Technology

Brigham Young University

November 2015

ABSTRACT

Deriving System Vulnerabilities
Using Log Analytics

Matthew Somers Higbee
School of Technology, BYU
Master of Science

System Administrators use many of the same tactics that are implemented by hackers to validate the security of their systems, such as port scanning and vulnerability scanning. Port scanning is slow, and can be highly inaccurate. After a scan is complete, the results of the scan must be cross checked with a vulnerability database to discover if any vulnerabilities are present.

While these techniques are useful, they have severe limitations. System Administrators have full access to all of their machines. They should not have to rely exclusively on port scanning them from the outside of their machines to check for vulnerabilities when they have this level of access. This thesis introduces a novel concept for replacing port scanning with a Log File Inventory Management System. This system will be able to automatically build an accurate system inventory using existing log files. This system inventory will then be automatically cross checked with a database of known vulnerabilities in real-time resulting in faster and more accurate vulnerability reporting than is found in traditional port scanning methods.

Keywords: log file, agent, syslog, elasticsearch, logstash, kibana, software inventory, inventory management, vulnerability, port scan

ACKNOWLEDGEMENTS

I would like to thank Dale Rowe, Derek Hansen, and J. Ekstom for their support and for mentoring me through the research and writing process. A big thanks to the SEA organization for sharing their data. I would like to express my gratitude towards Lane Broadbent and Hans Farnback for sharing their expertise of ELK, for helping me overcome some technical problems, and for being my liaison with SEA. Finally, I would like to thank my wife, Daphne, for always being there for me, and for encouraging me when things were tough.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Nature of the Problem

Despite having Administrative access to their organization's machines, many System Administrators must rely on the same tools and procedures employed by malicious hackers in order to verify and validate the overall security status of their equipment. A common technique used to gauge the overall security of any network-connected device is to cross check the results of a port scan with a vulnerability database. While this technique can be highly effective for gathering data about a system with the intent to exploit it, the inherent limitations of port scanning can make it difficult for System Administrators to keep their systems secure.

Port scanning is slow, and can be inaccurate depending on the current state of the device. It can take several hours to fully scan just a single server. Port scans attempt to fingerprint (or guess) the services running on each open port along with the Operating System that is running on the machine. The accuracy of these fingerprints vary, and in some cases the services cannot be fingerprinted at all. For example, some software packages do not perform network operations, meaning that they cannot be fingerprinted, yet they are still vulnerable to a local privilege attack. Administrators often have to manually start the port scanning process, which can result in infrequent scans that do not follow a set schedule. If a service is temporarily down during the scan, the results of that scan will not give a fully accurate representation of the system.

Another method of defense is for System Administrators to run services on different ports than normal. This can make it difficult for attackers to know what services are running on which ports. However, this method can also make it difficult for System Administrators to accurately gauge their systems through port scanning techniques. Port scanning also uses various techniques to guess what Operating System or firmware is running on the machine, along with what ports are open, what services are running on those ports, and versions of those services. These methods are never completely accurate, and the level of accuracy varies from platform to platform.

System Administrators have full access to their machines. They should not have to rely on port scanning them from outside to check for vulnerabilities. This is a good method for them to see what potential attackers can see, but it is a poor tool for them to actually lockdown their equipment.

1.2 Purpose of the Research

The purpose of this research is to assess the viability of using a log file based inventory management system to cross check devices with a vulnerability database and to ascertain the benefits and limitations that such an approach would bring.

1.3 Project Approach

The novel aspects of this research are to develop and evaluate new techniques to build and update a system inventory in real-time through the use of log files, develop and evaluate a standard to gather log file data with the intent to cross check all relevant data with a vulnerability database, and to develop a conceptual framework for gathering the data from log files and to analyze the data. In the situation in which existing log files are not sufficient, other methods will

need to be considered to gather the requisite data, such as creating custom agents to gather the appropriate data and generate logs suitable for the proposed system.

Log files will be gathered from several devices to inventory the software running on them. That data will be sent to a centralized server to build the inventory and to cross check it with known vulnerabilities all in real-time. Figure 1 shows the basic architecture for this approach.

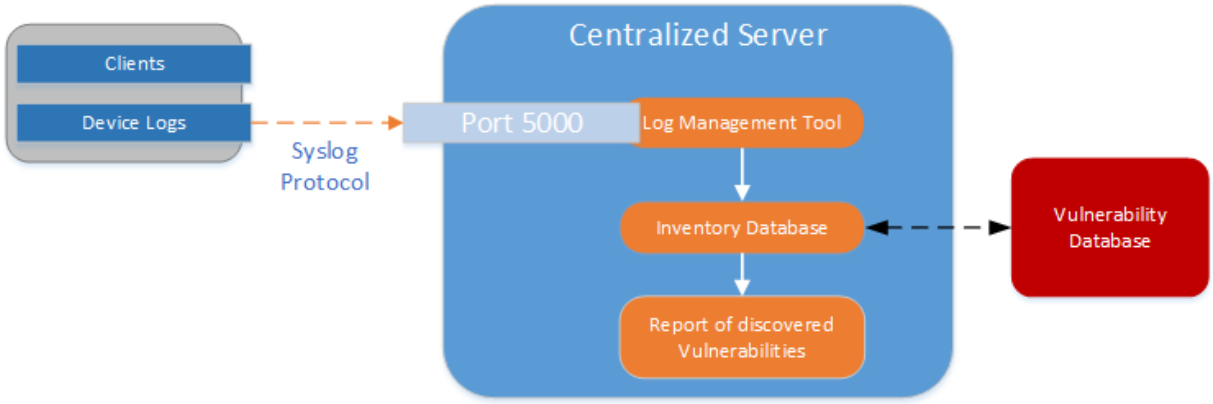

Figure 1: Architecture for Log File Inventory Management System

1.4 Scope and Limitations

The scope of this research will be limited to the reported information from existing or custom log files for a specific set of defined devices. Only a limited subset of existing Operating Systems, applications, and updates will be used. Likewise, the scope will also be limited to using an existing vulnerability database instead of creating a customized one. In the event that no existing vulnerability databases contain the required information in a usable format, a custom vulnerability database will be manually compiled using other databases.

1.5 Research Questions and Hypotheses

The following questions will be answered from this research:

- (R1) How does the log file inventory management system compare with traditional port scanning with regards to speed and accuracy?
- (R2) What are the challenges of creating a log file inventory management system and what techniques can be used to overcome them?
- (R3) What are the key indicators required from log files to cross check them with a vulnerability database?

The following hypotheses will be tested as part of this research:

- (H1) It is possible to generate an accurate system inventory using log files.
- (H2) It is possible to generate an accurate inventory of vulnerabilities that may exist in the current configuration.
- (H3) The log file inventory management system will enable faster alerting of vulnerabilities than port scanning.

1.6 Definitions

- Agent Technology – A primitive form of artificial intelligence that is able to recognize an environment and respond to alerts (Kercher 2013).
- Log File – A computer file in which a program records events, such as user access or data manipulation as they occur, to serve as an audit trail, diagnostic device, or security measure.
- Port Scanner – A software application designed to probe a host for open ports.

- Vulnerability Database – A collection of known software vulnerabilities often with a score to represent the severity of the vulnerability.

1.7 Thesis Outline

The remainder of this thesis will be outlined as follows:

1.7.1 Chapter 2: Literature review

This chapter contains a comprehensive study of existing and related research to the research questions and hypotheses listed above.

1.7.2 Chapter 3: Methodology

This chapter explains how the research will be carried out and how the data was gathered and analyzed.

1.7.3 Chapter 4: Framework

This chapter contains a detailed discussion of everything that was done with regards to this research including all information about how relevant data was generated.

1.7.4 Chapter 5: Results

This chapter contains an in depth analysis of the data and its significance to the research field.

1.7.5 Chapter 6: Conclusions and Future Work

This chapter contains conclusions regarding the validity of the previously mentioned research questions along with general insights.

1.7.6 Appendix A. ELK Configuration Files

This Appendix contains the contents of all of the configuration files used to parse, store, analyze, and visualize the data using the ELK stack.

1.7.7 Appendix B. MySQL Database Configuration

This Appendix contains an SQL script that can be used to recreate the schema of the tables that were used as part of this research.

1.7.8 Appendix C. Python Scripts

This Appendix contains the Python scripts that were used to compare the log file data with known vulnerabilities.

1.7.9 Appendix D. Comparison for Windows 7

This Appendix contains the test results for Windows 7.

2 LITERATURE REVIEW

2.1 Introduction

This chapter examines existing research related to creating a log file inventory management system, using a standard for comparing log files with a vulnerability database, performing centralized log file correlation, relevant syslog and agent technologies, and supporting situational awareness. Combining all of these components is required in order to answer the questions presented in the prior chapter.

2.2 Vulnerability

A vulnerability is a weakness or flaw in computer software that can allow attackers to gain some level of unauthorized access to a machine or to information stored on a machine. The severity of the vulnerability is directly correlated with the level of unauthorized access an attacker can obtain by exploiting the vulnerability.

Vulnerabilities in software are usually found in firmware, Operating Systems, or in the applications that run on those Operating Systems. Due to the sheer number of applications and Operating Systems, it can be difficult to know whether or not the code running on any given machine is vulnerable to outside attack.

Two examples of recent major vulnerabilities are Heartbleed and Shellshock. Both of these vulnerabilities were discovered in 2014. Heartbleed involved an improper input validation in the

OpenSSL cryptography library which is heavily used in the implementation of the Transport Layer Security protocol (Canada 2014). By exploiting this vulnerability, attackers could gain access to users' session cookies and passwords, and servers' private keys.

The Shellshock vulnerability was a bug within the Bash shell. Many public facing services such as web servers use Bash to process certain requests. Shellshock allows attackers to exploit vulnerable versions of Bash to execute arbitrary commands meaning that they could gain unauthorized access to the entire computer system.

Several organizations have created databases of known vulnerabilities such as Heartbleed or Shellshock. These databases provide information regarding the vulnerabilities such as which version of an application is affected or what can be compromised by a successful exploit. However, these databases often are inconsistent with one another with respects to naming conventions, actual data stored, and format. Additionally, the entries in the databases are also often inconsistent with regards to what data is stored. These inconsistencies can make it difficult to automate a process to check these databases(Mell, Scarfone, and Romanosky 2006).

2.3 Vulnerability Database

There are several existing vulnerability databases. These databases differ in content and format, but all try to provide critical information about known vulnerabilities. Some of the more popular databases include:

- National Vulnerability Database (NVD) – A U.S. government repository of standards based vulnerability management data (NVD 2015). The entire database is available for free download in XML format.

- Open Source Vulnerability Database (OSVDB) – An open source web based vulnerability database. The entire database can be queried using their website, but users are limited to two queries a day using their API (OSVDB 2015). The database is not available for download.
- United States Computer Emergency Readiness Team (US-CERT) – A government team that attempts to improve the nation's cyber security (US-CERT 2015). This website offers RSS feeds and alerts to keep users updated on current cyber security issues.
- Vendor Databases – Several vendor specific databases exist such as Microsoft's Security Bulletin which provides information on vulnerabilities specific to Microsoft products.

Because the National Vulnerability Database is comprehensive, free, and available for download it will be used as the primary database for this research.

2.4 Port Scanning

A port scan is a method to discover what services are running on a machine. The scan iterates over all of the specified ports on a machine looking for open ones. When the scan discovers that a port is open it attempts to determine what service is running on that port, and what version of that service is installed. Based off of this information, intruders can attempt to exploit the machine, and system administrators can attempt to patch any discovered vulnerabilities (Teo 2000). Port scanning can pose a stability risk to software. Port scans can cause software to crash or malfunction. Port scans are also frequently blocked by firewalls and Intrusion Detection Systems (IDS). This can limit the usefulness of running port scans. However,

many of the devices that are at risk to port scans generate log files, and support the syslog protocol.

2.5 Log Files

The inner workings of computers and applications can be difficult for developers and System Administrators to understand. This can make it hard for them to understand why things are not working the way they think they should. In order to alleviate this problem, programmers have set up a system to write events to a log file. These files can provide highly useful insights into system operation and to debug undesired computing results. Log files also usually contain identifying information regarding the application or service that is writing to a log file. This information is not always found on every log entry, but it is often included in certain entries such as during the reboot cycle or after installing updates. This information can be useful to identifying what services are running on a machine. For example, one Windows Event Log includes the following information: Microsoft (R) Windows (R) 6.0.1 7601 Service Pack 1, where Windows 6.01 represents Windows 7.

Typically, these log files are not written to a standard format. Rather they are customized to record the events and information that is important to their services. This can make it difficult to gather and analyze information from multiple services (Havens 2011).

2.6 Syslog

In the 1980's Eric Allman created a logging standard known as syslog. This standard includes the basic layout for a log entry along with a protocol to transmit logs to a logging server. Since its creation, this protocol has been improved by defining the standards it uses more clearly,

and by adding additional transport and security documents to its definition (Havens 2011). This standard has become widely used for Linux systems and for many network devices such as printers and routers. Figure 2 shows a sample syslog message. The syslog payload consists of three major parts, the PRI, Header, and MSG which will be described below.

```
<86>Apr 10 10:37:01 debian-higbee sshd[23757]: pam_unix(sshd:session): session opened for user root by (uid=0)
```

Figure 2: Example of Syslog

2.6.1 PRI

The PRI portion of the syslog is meant to describe both the priority and facility of the log entry. The PRI consists of 1 to 3 digits enclosed inside of two angle brackets (<>). Table 1 shows the potential facility values and Table 2 shows the potential values for the Priority Field.

Table 1: Numeric Codes for Facility Field (RFC 3164)

Numerical Code	*Facility*
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security / authorization messages
5	internal syslogd messages
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	security / authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit

Table 1 Contd.

14	log alert
15	clock daemon
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

Table 2: Numeric Code for Priority Field (RFC 3164):

Numerical Code	*Priority*
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

2.6.2 Header

The header contains a timestamp followed by a single space, and then the hostname of the machine the log originated from. If that machine does not have a hostname the IP address of that machine may be sent instead.

2.6.3 MSG

The MSG section contains the name of the source program that generated the log, plus the actual contents of the log. The MSG section has a loose structure that can allow different

syslog messages to have slightly different formats, which can make it difficult to parse through all of them.

2.6.4 Network

Typically, syslog messages are sent over UDP through port 514. Some administrators prefer the reliability of TCP to the speed of UDP and so syslog has evolved to support both protocols. Syslog also supports the option to mirror or relay messages to other syslog servers allowing for a logging hierarchy to be established. One major drawback of Syslog is that it isn't protected. Anyone on the network could intercept and view the unencrypted packets.

2.7 Data Mining

Data Mining is the process of identifying patterns within large datasets by applying concepts from artificial intelligence, machine learning, and database systems. The basic principle of data mining is to extract information from a data set and transform it into something understandable and actionable. Data Analysis systems typically consist of 5 major components:

- A Shipper program, which sends log files from clients to a centralized server
- A Broker program, which aggregates the data and queues it until it is ingested
- An Indexer program, which breaks the data up into individual searchable fields
- A Search and Storage program, stores the data and indexes it to facilitate searching
- An Interface to allow users to interact with the stored data.

By combining these components, it is possible to mine data from log files and turn it into useful information (Lund et al. 2015).

2.8 Agent Technology

An agent is a primitive intelligence that is able to perform autonomous action in an environment in order to meet programmed objectives (Kercher 2013). An agent is used to observe its environment and identify conditions to act upon (Wooldridge 1998). Agents will be used to provide custom log files for necessary data that is not already incorporated into existing syslog files. For example, an agent installed on a Windows machine could be designed to report on information pertaining to a third party application, such as a browser, which is not typically included inside of Windows log files.

2.9 Simple Network Management Protocol

Another technology that is used to monitor and manage network devices is the Simple Networking Management Protocol (SNMP). This protocol typically supported by routers, switches, printers, servers, workstations, and other devices. This protocol uses agents running on the devices, often known as managed devices, to report information to a manager or server. Through these agents the server can gather information about all the managed devices, and conduct active tasks such as modify configuration files. There have been three major versions of SNMP, and the latest version addresses many security concerns regarding the protocol, but may still be vulnerable to brute force and dictionary attacks when weak authentication keys are used (RFC1098 2015).

2.10 Log File Inventory Management System

While very little work has been done previously to validate the use of log files in creating a system inventory, the components of such a system have been researched and discussed in great detail. The three major components of a Log File Inventory Management System include:

configuration management, centralized log file correlation, and log management for incident response.

2.10.1 Configuration Management

Configuration Management is a process for establishing and maintaining the consistency of a product. For a Log File Inventory System to work effectively, it will need to incorporate several concepts from Configuration Management. Traditionally Configuration Management is comprised of four activities: identification, control, status, and audit (Bendix and Pendleton 2013).

- Configuration Identification – The act of identifying all of the important parts that should be put under configuration control.
- Configuration Control – The act of managing changes made to the configurations.
- Configuration Status – The act of providing a current status of the configurations.
- Configuration Audit – A sanity check to ensure that it will deliver as promised.

The proposed solution in particular will incorporate the concept of Configuration Status and Configuration Audit. Configuration Identification and Control will not be used as those aspects are needed when creating an all-encompassing adaptable product. Since this research will be used to generate a proof of concept prototype only specified configurations will need to be created and audited.

2.10.2 Centralized Log File Correlation

Centralized Log File Correlation is the process of using a centralized hub for log file analysis. This is especially useful for a system that gathers log files from more than one device. There are five steps that have been identified that need be followed in order to implement this

central hub effectively: centralization, normalization, consolidation, aggregation and correlation (Rinnan 2005).

- Centralization – Gathering log files from multiple devices into one place.
- Normalization – The process of reducing a complex data structure into its simplest form.
- Consolidation – The process of combining data from different formats into one unified view.
- Aggregation – The process of grouping distinct data.
- Correlation – The process of analyzing the data in real-time.

The proposed solution will utilize four of these concepts. Log files will be sent to a centralized server. This server will normalize and consolidate the data. The data will then be analyzed in real time and compared against a vulnerability database.

2.10.3 Log Management for Incident Response

Incident response refers to an organized approach to managing the aftereffects of a security breach or attack. In order for an incident response to be effective, there are a few key characteristics that log files should include. These characteristics are also important for an inventory system in order to ensure the authenticity of the log files themselves. These characteristics are: integrity, time stamping, and data reduction (Forte 2005).

- Integrity – The log files must not be altered in any way by an unauthorized person or program.
- Time Stamping – The logs must be time stamped in order to review and analyze them accurately later.

- Data Reduction – It is important to be able to reduce the log files down the key data in order to quickly and accurately analyze them.

The proposed solution will incorporate two of these ideas. The logs will be time stamped to ensure that they can be accurately analyzed. Likewise, the data will be reduced down to the key indicators so that they can be quickly and accurately analyzed. Integrity is important for a production level product. Since this solution will involve creating a proof of concept prototype, that aspect will not be included.

Data Integrity can be guaranteed through the use of a hashing algorithm. A hash function maps data of arbitrary size into a fixed size. The hashed value is uniquely generated based on the original data. Passing a hashed value along with the original data allows the centralized server to create its own hashed value of the original data that it received and compare it with the hash that was sent with the data. If the hashes match, then the data was not altered during transit. Note, that while it is mathematically possible to generate the same hash value from different data, the odds of it happening are so slim that hashing is generally accepted as a method to prove data integrity.

3 METHODOLOGY

This chapter outlines how the research will be conducted and how answers for the proposed research questions will be found.

3.1 (R1) Log File Inventory Management System

The purpose of this research is to assess the viability of using a log file based inventory management system to cross check devices with a vulnerability database. It was determined not to use SNMP for the following three reasons. 1) It has been done before, 2) SNMP doesn't include the same real-time benefits that come from using log files, and 3) because this research is attempting to find a one-fits-all solution instead of using several different technologies for different situations. To examine the effectiveness of this approach, a proof of concept prototype will be built and its performance will be compared with that of traditional port scanning methods. The development process of this prototype will inherently verify what components are necessary to create the Log File Inventory Management System.

The development phase will consist of three parts, client side syslog/agents, a centralized log file analysis server, and cross checking the log files with a vulnerability database. The syslog/agents monitor system and application configurations and send identifying information to the centralized server. The server analyzes the log files in real time and cross checks the identifying information with known vulnerabilities and reports any vulnerabilities currently

present on the clients. Figure 3 is a flowchart that details how the clients will interact with the centralized logging server.

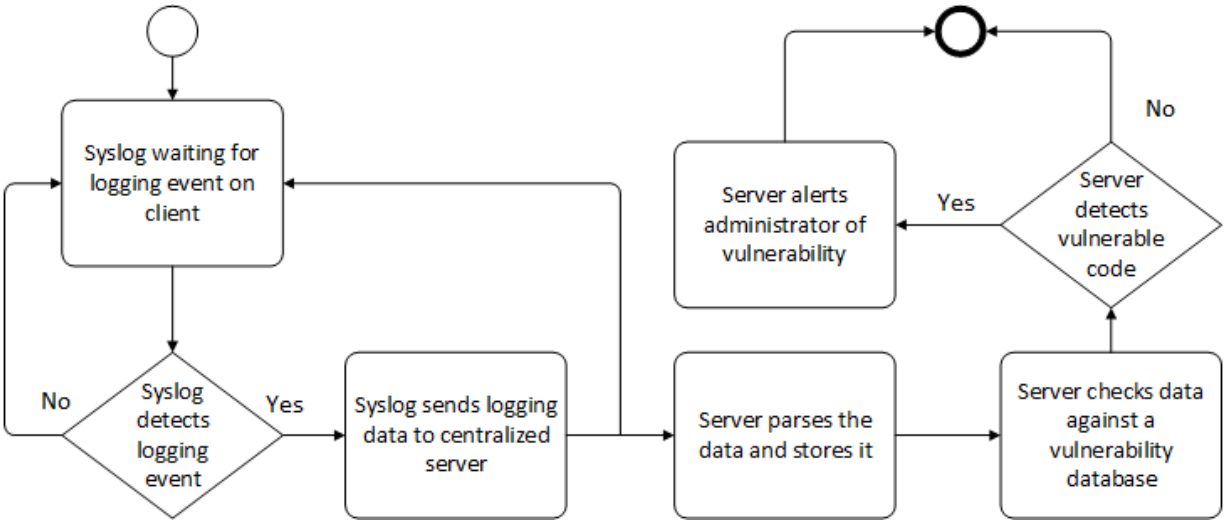

Figure 3: Prototype Flowchart

3.1.1 Prototype

Linux based systems, and many network devices such as printers and firewalls support the syslog format natively. However, Microsoft systems do not natively support the syslog standard. Several programs exist to "translate" Microsoft Event Logs into the syslog format. In order to simplify the process for parsing the log files, one of these translation programs will be installed on the Windows machine to make all of the log files conform to the syslog protocol. This program will need to be open source, configurable, easy to use, and readily available.

The clients will be configured to send syslog information to a centralized server. In the case where necessary information is not contained in the syslog files, custom agents will be written and deployed to gather and transmit the requisite data wherever possible. Any required

data that is not already part of existing log files will be identified during the development of this prototype.

The centralized server will need to use a log management tool to capture incoming syslog messages, process and parse the messages, and store the messages for further analysis. This tool will need to provide some method for external access to the stored data such as an API or it will need to use a database that has functionality built in for external access such as a SQL database. The data will need to be stored in real-time and will also need to be accessible in real-time. This tool will also need to be open source, well documented, and highly customizable. It will need to have the ability to listen on multiple ports for messages using both TCP and UDP. It will also need to incorporate the ability to transform and normalize the data from the syslog messages before storing it.

The identifying information from the log files will then be checked against a vulnerability database. Several databases exist, each formatted differently. A few of the databases are explained in greater detail in Section 2.3. The database(s) that will be used for this research will be selected during the implementation phase after testing them to ensure that they will be suitable. The database(s) will need to be downloadable, easy to parse, complete, and will need to contain all relevant fields such as Severity and Affects Version. The overall topology for the Log File Inventory Management System is found in Figure 4.

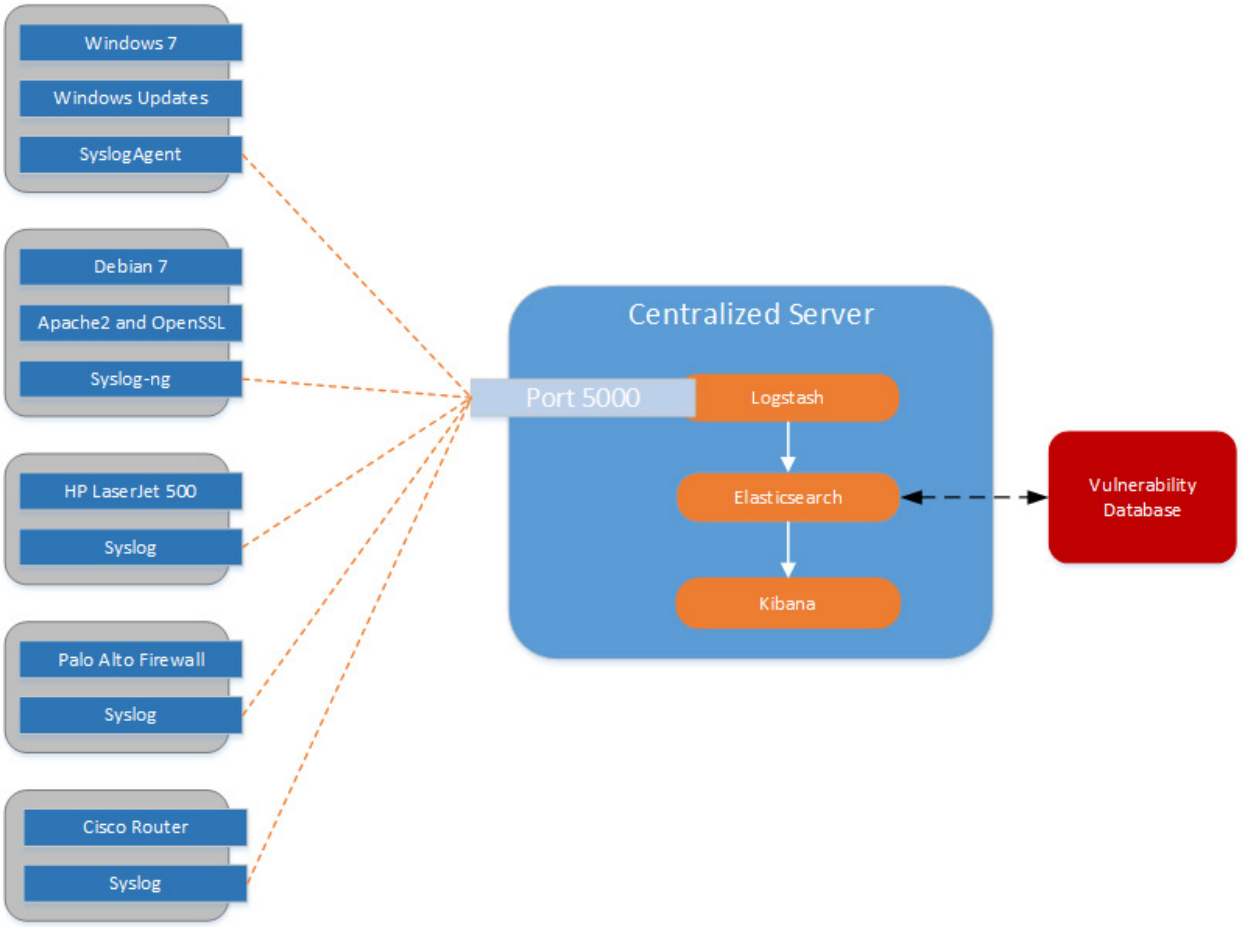

Figure 4: Proposed Log File Inventory Management System

3.2 (R2) Overcoming Challenges of Log File Inventory Management System

The challenges of implementing a Log File Inventory Management System will be identified during the course of creating the prototype. Notes on these challenges will be captured by the author on a frequent basis throughout the development process. The solutions to these challenges will be incorporated into the prototype to provide the required functionality. Specific technical challenges will be classified into more general problem types with this approach.

3.3 (R3) Log File Key Indicators

To effectively compare the log files with a vulnerability database, several key indicators will need to be identified. These indicators will be used to check the current status of any given device with known vulnerabilities. During the course of the creation of the prototype, these indicators will be determined. They will consist of the necessary core elements that are required to compare the originating system of the log files with known vulnerabilities.

3.4 (H1) It is Possible to Generate an Accurate System Inventory using Log Files

This hypothesis will be tested by automatically generating a system inventory using log files from network devices and manually comparing the results with the network devices. This will primarily consist of manually analyzing the firmware, Operating System, and updates installed on each device to ensure that the software running on each device was accurately depicted in the log files.

3.5 (H2) It is Possible to Accurately Detect Vulnerabilities using Log Files

The data for this hypothesis will be gathered by compiling a list of all vulnerable software from the devices using their log files and a vulnerability database. These will then be manually compared with the software running on the devices to ensure determine if the vulnerabilities actually exist on the machines. The information will also be checked to see if the vulnerabilities were not reported that should have been.

3.6 (H3) Log Files can Enable Faster Vulnerability Alerts than Port Scanning

There are two metrics for speed for testing this hypothesis. The first metric deals with how much time it takes for the vulnerability assessment to be started after there is a change in the

software configuration. Port scans must be manually started, or automated to run at specific intervals. The log file method will run in real-time when changes are made to the software on the machine. The log file method is almost always going to be faster in this metric because of its inherent real-time aspect. It is difficult to compare synchronous and asynchronous methods in terms of speed. In order to do this, specific intervals for the port scan will be specified, such as daily, weekly, and monthly. The prototype will then be compared with these intervals to determine the difference in speed.

The second metric for speed is how long it takes to actually conduct the vulnerability analysis. For port scanning that means how long does it take to report a vulnerability from the time the scan is first initiated. For the log file method that means how long does it take to report a vulnerability from the time the entry is first written to the log file.

This assessment of both tools will be done to see if there are quantifiable benefits to using the log file method over port scanning in terms of the speed with which they conduct the vulnerability assessment.

3.6.1 Data Collection

The prototype will monitor the log files from the following proposed devices:

- Windows 7
- Debian 7 with Apache webserver
- HP LaserJet 500 printer
- Palo Alto VM100 firewall
- Cisco 3500 router

The proposed devices were chosen to represent a wide range of commonly used devices and Operating Systems. They represent a large proportion of devices in use today. Windows 7 was proposed because it is one of the most commonly used Microsoft operating systems currently. Debian 7 with apache was proposed to represent the Linux distributions since it is widely used, and it is the latest stable release. The printer, firewall, and router were all proposed because they represent commonly used devices, and the Information Technology program at BYU already has access to all of them. This wide sampling will be used as part of the prototype to help validate the benefits of using the log file method.

The following identifying data will be gathered from the log files from the devices proposed above to build an accurate inventory of the software running on them:

- Operating System / Firmware
- Service Packs / Updates
- Running services identified previously
- Version of running service
- Additional data as is deemed necessary

This data will then be cross checked with known vulnerabilities, and the results will be compared to the results of using a port scan.

4 FRAMEWORK

There are two major components to the proposed framework, the centralized log file server and the client side configurations. The following sections detail the implementation of these framework components.

4.1 Centralized Log File Data Collection Server

The server consists of a Debian 7 Virtual Machine running on the Brigham Young University Cyber Security and Research Lab network. Debian 7 was chosen to run on the central server because it is a commonly used distribution of Linux and it is the latest stable release of Debian,.

4.1.1 ELK Configuration

The centralized server will use Elasticsearch, Logstash, and Kibana (the ELK stack) to gather and analyze log files. The ELK stack is an open source real time search and analytics engine. Elasticsearch also provides an API to access the log file information stored there making it a feasible process to compare the data in real time with a vulnerability database. Logstash stores log files into Elasticsearch and additionally allows users to transform the data using filters and parsing algorithms. Kibana is a visualization engine for Elasticsearch that allows users to interact with their data through custom web based dashboards. Combined these programs

provide a powerful log management capability. The ELK stack was chosen for this prototype because it is open source, reliable, highly customizable, and is widely used.

The server uses Elasticsearch 1.4.2, Logstash 1.4.2, and Kibana 3.1.2 to gather, parse, and store syslog events transmitted over the network from the clients. These versions were chosen because they were the latest versions at the beginning of the initial research. It is also running MySQL to store the vulnerability databases, and uses Python 2.7 to compare the data stored in Elasticsearch with the vulnerabilities in the MySQL database. Table 3 shows Python Libraries that were used.

Table 3: Python Libraries

Library Name	*Description*
urllib2	Used to open and read URL requests
json	Used to parse JSON objects
datetime	Used to perform operations on dates and times
pymysql	Used to connect to a MySQL database and execute commands
re	Used to support regular expressions
requests	Used to add base64 password encoding to URL request
base64	Used to encode a URL password to access the SEA database
socket	Used to send data into ELK

Logstash listens on port 5000 for any incoming UDP or TCP traffic. It then uses a custom configuration file to specifically look for syslog transmissions and parse them into separate fields that can be used to filter the data. The syslog messages do not all conform to the exact same specifications so each field isn't necessarily present in each syslog transmission. Table 4 shows the fields that are being parsed by the configuration file. After parsing the data Logstash stores it inside of Elasticsearch.

Table 4: Parsed Fields from Syslog

Field	*Description*
syslog_pri	A numerical value representing the priority and facility of the message.
syslog_timestamp	A timestamp
syslog_hostname	The hostname or IP address of the device where the log file originated
syslog_program	The program that sent the syslog
syslog_pid	The ID of the program that sent the syslog
syslog_program_version	The version of the program that sent the syslog
syslog_message	The message contents of the syslog

4.1.2 Elasticsearch API

Elasticsearch provides a built in REST API that allows external access to the data stored there. This API allows programs to GET, DELETE, or UPDATE the data stored in Elasticsearch individually or to GET the data in bulk. By default, the GET method is in real-time and is unaffected by the index refresh rate. Built on top of this, is the Elasticsearch Query DSL, which is based on JSON and is used to define queries to retrieve the desired data. The proof-of-concept prototype uses these technologies built into Elasticsearch to retrieve the identifying information from the log files to compare them with the vulnerability database.

4.1.3 Python Script

A python script (see Appendix C.) compares the identifying information being stored into Elasticsearch with the vulnerability data stored in a local MySQL database. When the script discovers a known potential vulnerability, it sends a message containing the name of the vulnerability, the severity of the vulnerability, the CVE of the vulnerability, and the IP address of the vulnerable machine into ELK. Logstash parses the message and stores it into Elasticsearch, while Kibana displays a graph showing the discovered vulnerabilities. It was determined to use

ELK to visualize and report discovered vulnerabilities because it allowed for one simple user interface (Kibana) to be used. Through ELK, it is possible to see what is running on the machines in the network, a high level overview of the discovered vulnerabilities, and the option to drill down and get specific vulnerability data.

4.1.4 Vulnerability Database

Initially it was believed that the National Vulnerability Database would be ideal for cross checking configurations for known vulnerabilities. This was determined based on the wide range of platforms and vulnerabilities that are contained within the National Vulnerability Database. However, the structure of the database is very loose and is not conducive for parsing out the requisite data. There is no good way to see each version of software that is affected by a vulnerability in the NVD. The problem is derived from the fact that the database uses a single field (description) to contain both description and affected versions. This can lead to difficulties when parsing the data.

- Sometimes more than one software is listed in the description. Searching for both the program and version can result in false positives. For example, searching for Apache 2.2.22 can result in an entry that mentions Apache and version 2.2.22 of a completely unrelated software.
- If a vulnerability affects multiple versions of the software, there is no standard way for listing them. For example, the description could say that it affects all versions before 2.2.4, or that it affects 2.2.x, or that it affects 2.2.1-4.

These limitations make the NVD a very impractical choice for using it to cross check the client configurations for known vulnerabilities.

Similarly, the Open Source Vulnerability Database (OSVDB), Mitre Common Vulnerabilities and Exposures (CVE), and the United States Computer Emergency Readiness Team (US-CERT) databases were determined to be an impractical choice for the same reasons as the NVD.

After further research it was decided to use vendor specific vulnerability databases as much as possible. The Microsoft Security Bulletin was selected for all Microsoft vulnerabilities. The Microsoft Security Bulletin is designed in such a way to be highly conducive for checking for vulnerabilities. The Security Bulletin is structured in such a way to give it three distinct advantages.

1. There is an entry in the database for each affected version. For example, if the same vulnerability affects Windows 7 and Windows 8, there may be four separate entries. One for Windows 7 32bit, Windows 7 64bit, Windows 8 32bit, and Windows 8 64bit. This makes it extremely easy to search for vulnerabilities.
2. Windows Security Updates each have a unique identifier in the format KB########. The Security Bulletin includes these identifiers in each vulnerability to show what update was issued to fix the problem. This means that the Centralized Server just needs to get all potential vulnerabilities for a client and see if the corresponding update has been applied. This approach results in very few false positives so long as the database is accurate because each vulnerability maps directly to an update, making it very easy to see if the vulnerability has been patched.
3. The Windows Security Updates are hierarchical. New updates might supersede a previous update if they are related. This means that clients may not need to install old updates if newer ones exist. The Security Bulletin tracks which updates supersede

others. This makes it is easy to ensure that a red flag is not raised because outdated updates have not been installed.

The Microsoft Security Bulletin could probably be optimized by normalizing it into a relational database, but it works well as it is for cross checking vulnerabilities in its provided format. The main limitation of the Security Bulletin is that it is limited to Microsoft only products.

Several problems were identified when attempting to use the vendor databases for open source technologies. Most of these databases are not downloadable, and do not have an external API. The data in them was also structured similarly to the NVD making them difficult to parse. Another issue is that the versioning of Linux packages is not consistent across platforms making it extremely difficult to automate a process for gathering vulnerabilities about these packages. This issue is best illustrated with the following workflow related to gathering the affects version and fix version for the Heartbleed vulnerability.

According to the CVE database all versions of OpenSSL between 1.0.1 and 1.0.1g are vulnerable to Heartbleed. However, according to the Debian Wheezy package database, the problem was fixed in the Debian version of OpenSSL, with the 1.0.1e-2+deb7u5 release. This is a direct conflict with the CVE database statement that all versions between 1.0.1 and 1.0.1g are vulnerable. To further illustrate this problem, the vulnerability was fixed in RedHat Linux 6 with version 1.0.1e-16.e16_5.7, also in contradiction to the CVE database.

Because of these limitations it was determined to remove the fully automated Linux vulnerability comparison from scope. Instead a few packages were selected and the vulnerability data was manually gathered and entered into the research database. This was done because the object of this research was to prove the viability of comparing log files with a vulnerability

database, not with being able to automate the creation of such a database. The Microsoft Security Bulletin along with the Windows client will serve as the primary proof-of-concept for a completely automated method of comparison while the open source technologies will be partially automated to prove that vulnerabilities can accurately be reported so long as the vulnerability data is stored efficiently.

Two Debian packages were selected for this comparison – Apache and OpenSSL. It was simple to test a vulnerable version of Apache since it was already included in the Debian 7 distribution and updates weren't installed. The apache log files were gathered sent to the centralized server using the Syslog protocol. Apache included the required identifying information in its log files each time someone accessed a website that it was serving. Several entries from Apache's vulnerability database were manually entered into the research database. (Apache 2015)

It was not a trivial matter to install a version of OpenSSL that was vulnerable to heartbleed. Because of the severity of heartbleed, the vulnerable versions were removed from the default packages so typical users could only install the updated versions. However, there is an organization that maintains snapshot.debian.org (Debian 2015) which is an archive containing packages from back as far as 2005. After finding the desired package on their website the following line of code was added to the sources.list file as shown in Figure 5:

```
deb http://snapshot.debian.org/archive/debian/20130211T215500Z/ testing main contrib non-free
```

Figure 5: Code Added to sources.list

Then running the following command shown in Figure 6 updated the package list and forced the package manager to allow the outdated repository.

```
Aptitude update –o Acquire::Check-Valid-Until=false
```

Figure 6: Update Package Manager

After running these commands, it was possible to install an outdated version of OpenSSL (1.0.1c-4) that is vulnerable to heartbleed. The vulnerability was manually entered into the database to allow the centralized server to compare the installed version of OpenSSL with the versions that are affected by heartbleed. Table 5 shows the extent to which each of the initial proposed devices found in section 3.6.1 were incorporated into the prototype.

Table 5: Usage of Proposed Devices

Platform	*Included in Prototype*	*Level of Automation*
Windows 7	Yes	Fully Automated
Debian 7	Yes	Partially Automated
Printer	No	N/A
Firewall	No	N/A
Switch	Yes	Partially Automated

4.1.5 MySQL Database

In order to better facilitate the comparison of the log files with multiple vulnerability databases it was decided to import all of the chosen vender databases into a custom MySQL database. This database not only contains a list of all of the known vulnerabilities within the

scope of the research, but it also stores the identifying information received from the log files of each host.

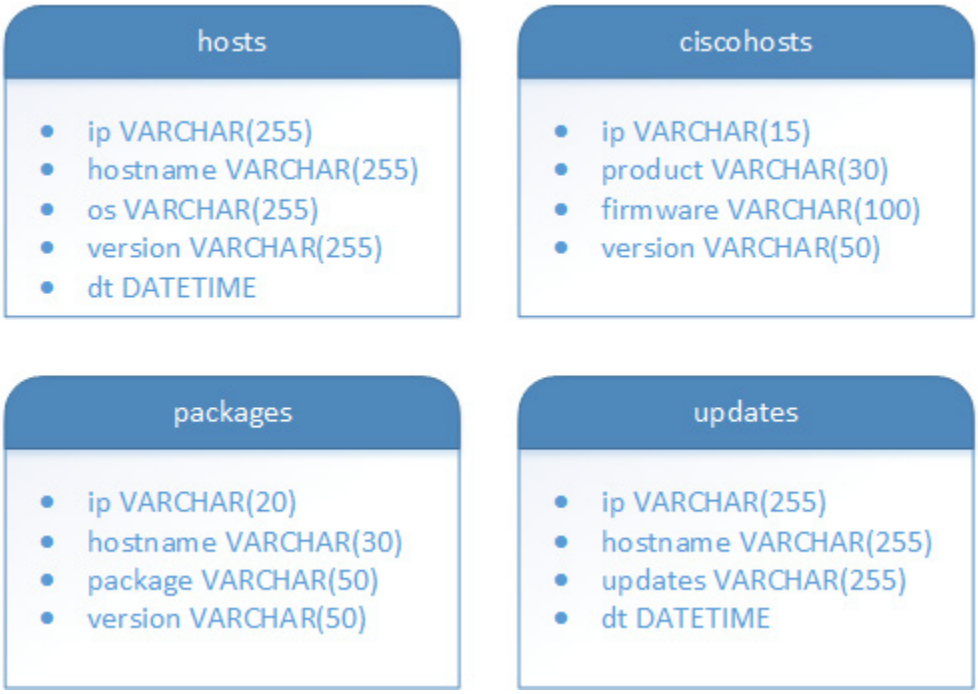

Figure 7: Database Schema for System Inventory

The database schema used to store the software inventory of the clients in the prototype is shown in Figure 7. The database stores information regarding the Windows, Linux, and Cisco systems in separate tables because they each track slightly different things. The *hosts* table tracks the basic information for each client regardless of vendor.

Figure 8 depicts the database schema that was used to store the lists of known vulnerabilities for each vendor. Because the vendors store significantly different types of information, the vulnerabilities were stored in separate tables as well.

Figure 8: Database Schema for Vulnerabilities

4.2 Client Side Configurations

Each client handles logging differently so it was necessary to configure each of them separately to transmit their log file data using the syslog protocol.

4.2.1 Windows 7

Windows does not support the syslog protocol natively, so Datagram SyslogAgent was installed to convert the event logs into the syslog format and transmit them to the centralized server. The Datagram SyslogAgent was chosen because it met the criteria found in section 3.1.1.

The agent was configured to send all of the System Event Log files which contain the requisite information to the centralized server over the port 5000 using UDP. Port 5000 was chosen over the standard port 514 in order to avoid the accidental insertion of data from a misconfigured client not in the research scope.

4.2.2 Debian 7

There are two major syslog programs for Linux, syslog-ng, and rsyslog. It was decided to use syslog-ng since it was already preinstalled. Syslog-ng was configured to send all of the aptitude installation logs, Apache error logs, along with the default logs to the Central Server over port 5000 using UDP.

4.2.3 Printer

The available printer was an HP LaserJet 500 color M551 running the 2305083_000200 firmware revision. However, log messages contained no usable information for this research even when rebooting or upgrading the firmware. Because of this it was determined to remove the printer from scope. It was determined that it would require more time to find a printer through trial and error that reported the requisite information than was justified by the objectives of this research. It was decided that the proof of concept prototype could be properly analyzed without using a printer. In fact, this drawback shows a real potential weakness for the completed prototype. It will not be compatible with any devices that are incapable of sending the requisite information.

4.2.4 Firewall

The available firewall was a Palo Alto 6.1.2 PA 4020. It was determined to remove the firewall from scope. Like the printer, this firewall didn't send any of the needed information in its log files even during a reboot or a firmware update. It was decided that removing the firewall from scope would be acceptable for the same reasons described above for the printer.

4.2.5 Switch

A group at BYU known as System Event Analytics (SEA) has conducted a significant amount of research by sending log files from devices all over campus to their ELK cluster. After requesting access, this data was made available for this research. The BYU Office of Information Technology (OIT) gathers syslog data from devices all around campus and stores that data into Event Tracker. SEA has arranged with them to mirror that traffic, and forward it to the SEA system. SEA then buffers the logs using Redis and stores it into ELK. This is illustrated in Figure 9 created by Lane Broadbent, BYU CT.

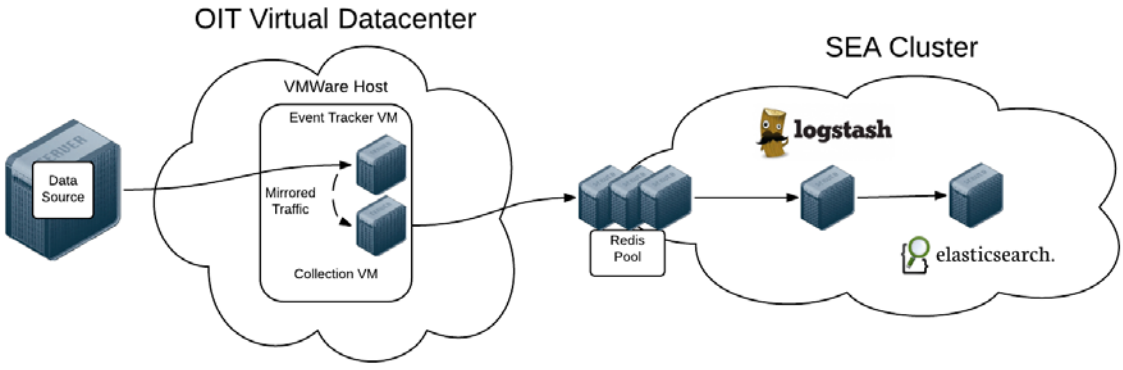

Figure 9: SEA Network Architecture

It is possible to view the logs generated by numerous Cisco switches by accessing the SEA instance of Elasticsearch. Due to the large pool of devices in their database it was determined to use the Cisco Catalyst 3560 log data for the purposes of this research. The Cisco Catalyst 3560 was chosen because there were recent log entries for this device in their database that indicated it may be possible to retrieve the necessary firmware information from them. Cisco maintains its own vulnerability database in cvrf format. Due to the complexity of this format it was decided to manually enter vulnerability information about the Cisco Catalyst 3560 into the local vulnerability database, instead of trying to parse the information out of Cisco's database automatically.

4.3 Problems Encountered with the Implementation

There were several problems that were encountered during the development phase of the prototype. The following sections explain the problems and illustrate the steps that were taken to overcome or circumvent them.

4.3.1 Loose Format of the Syslog Protocol

The Syslog protocol is notorious for having a loose standard. The MSG component of syslog does not have defined standard. This allows for several different devices to log different types of information depending on their requirements. This is useful, because it allows devices with different types of data to "mostly" conform to a single standard. However, this makes it difficult to parse all syslog entries in Logstash since messages may not all conform to the same exact standards. Because of this, several grok filters were created in the Logstash configuration files to account for the different Syslog messages that were being used. If additional syslog messages will be used in the future, it is likely that additional grok filters will need to be created.

4.3.2 Syslog Messages without Identifying Information from Network Devices

Some network devices do not contain the requisite identifying information in their syslog messages. For example, an HP printer was configured to send Syslog messages to the centralized server, yet those messages never contained information regarding what firmware was running on the printer. This is a real limitation for using the Log File Inventory Management System to actively monitor the security status of all network devices.

4.3.3 Kibana Reports Duplicate Vulnerabilities when Duplicate Logs Are Sent

Log files containing update or package installation data can be sent more than once from the same machine to the centralized server. Because the Python script reports a vulnerability back into Kibana every time one is discovered, or rediscovered, the Kibana interface may often report duplicate vulnerabilities. There are several recommendations that will be discussed in Chapter 6 that will help alleviate this problem.

4.3.4 Kibana is Designed to Show a History instead of the Current State

The dashboard interface in Kibana is configured to show data during a specific time interval. This makes it difficult to show the exact “Current State” of the devices. For example, a log file containing a vulnerability may have been sent 7 hours ago. If Kibana is configured to display vulnerabilities from the last 6 hours, the dashboard will not show the vulnerability despite the fact that it is still present on the machine.

Likewise, if a vulnerability is fixed within the interval being displayed on Kibana, Kibana will still show the vulnerability as being present, despite the fact that the vulnerability was fixed. Chapter 6 includes some recommendations to overcome these issues.

4.4 Testing

Developmental testing was conducted throughout the creation of the prototype to ensure that things were working as expected. The ELK server was tested to ensure that it was receiving Syslog messages and parsing them correctly. The Python script was tested to ensure that it was retrieving data from Elasticsearch, comparing that data with vulnerabilities in a MySQL database, and sending discovered vulnerabilities back into ELK. The clients were all tested to ensure that they were sending the requisite data via Syslog to the ELK server.

After development, the prototype was tested to assess the accuracy of the software running on the clients that was detected by the ELK server. The system was also tested to assess how much time it took to gather and parse log files, compare the data with the vulnerability database, report discovered vulnerabilities, and the accuracy of the vulnerabilities that were reported.

4.4.1 Summary of Test Results

Several key findings were discovered as a result of the testing and analysis that took place:

- The prototype was able to detect every installed update and package on the Windows and Linux Systems, as well as correctly identify the versions that were installed and the Operating Systems that were running on them.
- The prototype was able to detect with 100% accuracy when vulnerable software was installed on Linux, or when Windows was missing security updates.
- The prototype was able to report discovered vulnerabilities in real-time, often taking less than 30 seconds to do so from the moment the vulnerability was introduced to the system.

- The traditional port scan was unable to detect any installed updates on the default Windows 7 installation, or accurately guess the Operating System that was running.
- The traditional port scan did not detect any vulnerabilities. Instead it only detected the versions of a few pieces of software running on the machines. A user would have to cross reference this information with known vulnerabilities to see if the software was vulnerable.

Figure 10 shows the number of vulnerabilities that each method detected. The prototype detected all of them, while the port scan only detected one.

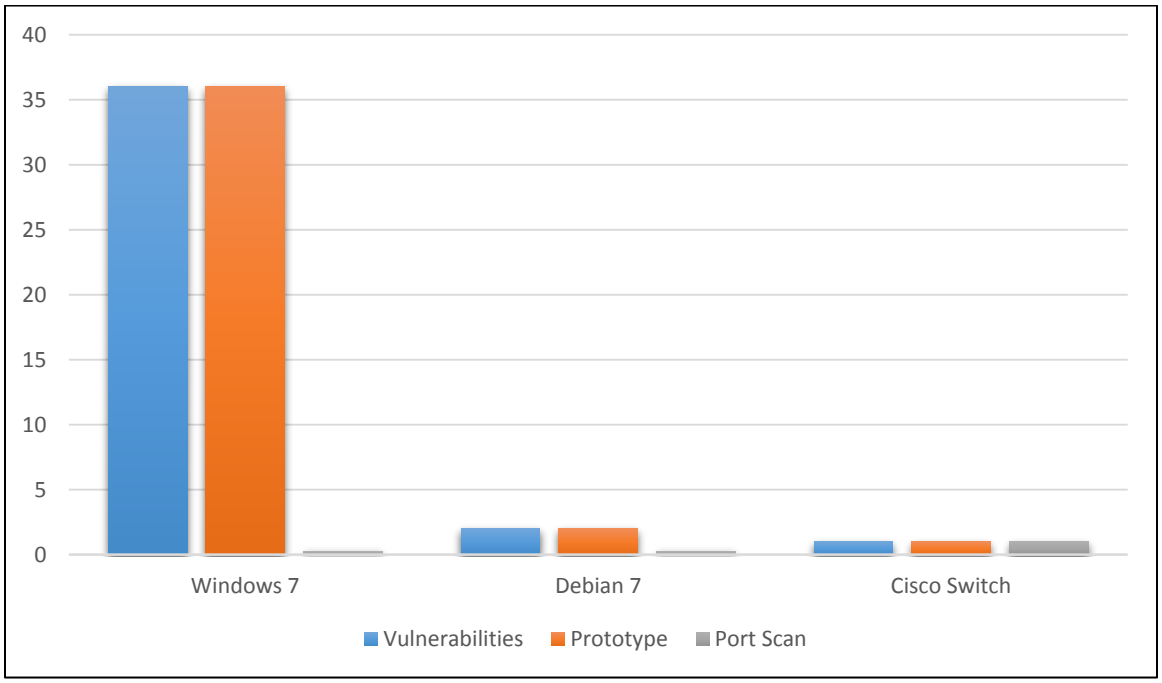

Figure 10: Comparison of Discovered Vulnerabilities

4.4.2 Accuracy of Software Inventory

To test the accuracy of the software detected by the ELK server, each client was manually inspected and compared with the results discovered by the ELK server. For the installed updates on Windows 7 the following command was run and compared with the data in the database that was gathered by the ELK server.

```
wmic afe list full /format:texttablewsys > "%USERPROFILE%\hotfix.txt
```

Figure 11: List Installed updates

This was also compared with the results of a port scan. There are several ways to configure a port scan, and the following command was used for each client.

```
nmap -T4 -A -v <IP Address>
```

Figure 12: Nmap Scan

Appendix D. Comparison for Windows 7 contains the results of the prototype compared with the port scan for Windows 7. The comparison shows that the research successfully created a completely accurate representation of the updates that were actually installed on the Windows 7 machine. The ELK server reported a complete and accurate representation of the OS, Architecture, and installed updates. The port scan, failed to detect any installed updates, reported the wrong OS, and didn't report anything for the architecture. The port scan was only able to detect that the following services were running: MSRPC, Microsoft-DS, cmrcservice, and ms-wbt-server. This shows that the prototype was much more complete and accurate than the port scan at detecting what updates were running on the machine.

Like many Linux distributions, Debian 7 comes with several packages preinstalled. None of these preinstalled packages were considered for this research. Instead, only the packages that were installed during the research period were recorded. To assess the accuracy of the packages installed on the Debian 7 machine, the apt log files were compared with the data stored in ELK. This was also compared with the results of running a port scan and is shown in Table 6.

Table 6: Comparison for Debian 7

	Debian 7 Manual Results	*Debian 7 Prototype Results*	*Debian 7 Port Scan*
IP Address	192.168.230.139	192.168.230.139	192.168.230.139
Operating System	Debian 7	Debian 7	Linux
Architecture	3.2.0-4-amd64	3.2.0-4-amd64	3.2 – 3.13
Installed Packages	Apache 2.2.22	Apache 2.2.22	Apache 2.2.22
	Slsh 2.2.4-15	Slsh 2.2.4-15	OpenSSH 6.0p1 4+deb7u2
	Libslang2-modules 2.2.4-15	Libslang2-modules 2.2.4-15	
	Jed-common 0.99.19-2.1	Jed-common 0.99.19-2.1	
	Jed 0.99.19-2.1	Jed 0.99.19-2.1	
	Libonig 2 5.9.1-1	Libonig 2 5.9.1-1	
	Openssl 1.0.1c-4	Openssl 1.0.1c-4	

The comparison shows that the prototype contained a completely accurate representation of the packages that were actually installed on the Debian 7 machine. However, the prototype did not report on any packages that were preinstalled such as OpenSSH. The port scan detected the correct version of Apache, and detected OpenSSH. This shows that the port scan can pick up things that the prototype cannot in the case where the packages were installed prior to the client being configured to send log files to the ELK server. This could be partially addressed in a future version of the prototype, by ingesting the results of the dpkg-query command when a Debian

client is initially introduced to the system. This command will generate a list of all installed packages.

However, the port scan missed several packages that were installed on the Debian machine, because they didn't have open ports associated with them. Also, the port scan was only able to determine that the client was Linux based. It couldn't tell which version of distribution or version of Linux was running. It also was only able to guess what kernel was installed 3.2 – 3.13. The installed kernel was included in the port scan's guess, but so were several other kernels. This shows that the prototype was more accurate than a port scan. Not only was the prototype more accurate, it was more complete. The port scan failed to detect the critical heartbleed vulnerability.

Since physical access to the equipment on SEA wasn't possible, the log files stored for the Cisco switches in their ELK cluster were compared the data stored in the ELK database used for this research. The results are shown in Table 7.

Table 7: Comparison for Cisco Switch

	Cisco Switch Manual Results	*Cisco Switch Prototype Results*	*Cisco Switch Port Scan*
IP Address	10.3.17.43	10.3.17.43	10.3.17.43
Firmware	IOS 15.0	IOS 15.0	IOS 15.0

The comparison shows that the ELK server used for this research contained a completely accurate representation of the firmware that was installed on the Cisco Catalyst 3560 Switch. It is also important to note that the port scan was similarly successful at detecting the firmware running on the cisco switch.

Even though the printer and firewall were removed from scope, it was decided to include a summary of a port scan of those devices to better show the data that the prototype was unable to retrieve. The port scan was able to detect that the printer was either an HP LaserJet 600 M602, HP LaserJet 500 M551, or a LaserJet M830. It did guess the correct model, but only with a 33% confidence level. The port scan inaccurately reported the Palo Alto firewall as a Linux 2.6.x device.

4.4.3 Speed to Report Vulnerabilities

It is difficult to compare the speed of the two methods because of the inherent difference in their natures. The prototype runs asynchronously in real-time, while the port scan runs synchronously at undefined intervals. Figure 13 15 illustrate the speed of reporting a vulnerability with port scans being conducted at different intervals. The Figures show a port scan being conducted daily at 8pm, weekly, and monthly on the first day of the month. Note that the time interval is not displayed to scale across the different figures.

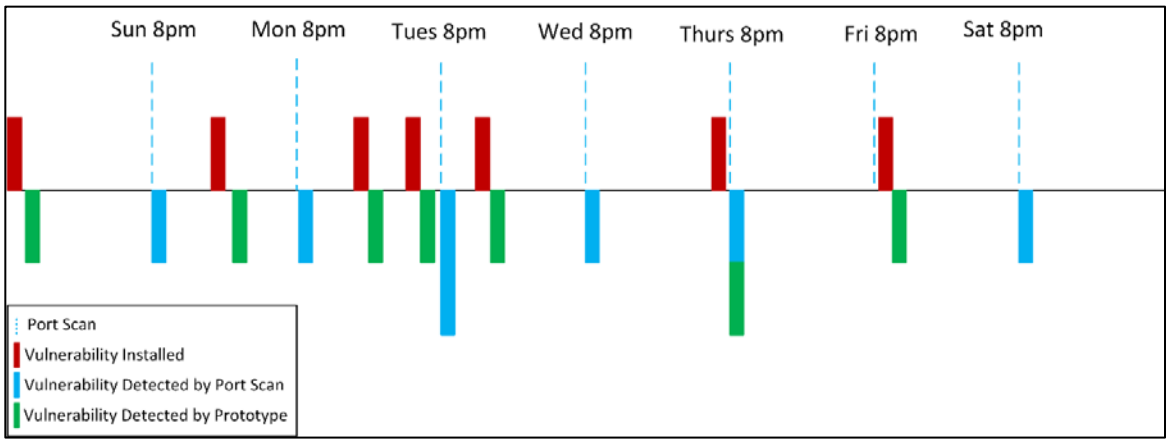

Figure 13: Speed Comparison for Daily Port Scan

As shown in this figure, the prototype will detect all vulnerabilities in real-time, but the port scan will detect them every 24 hours. This means that vulnerable software could be running for up to 24 hours before being detected.

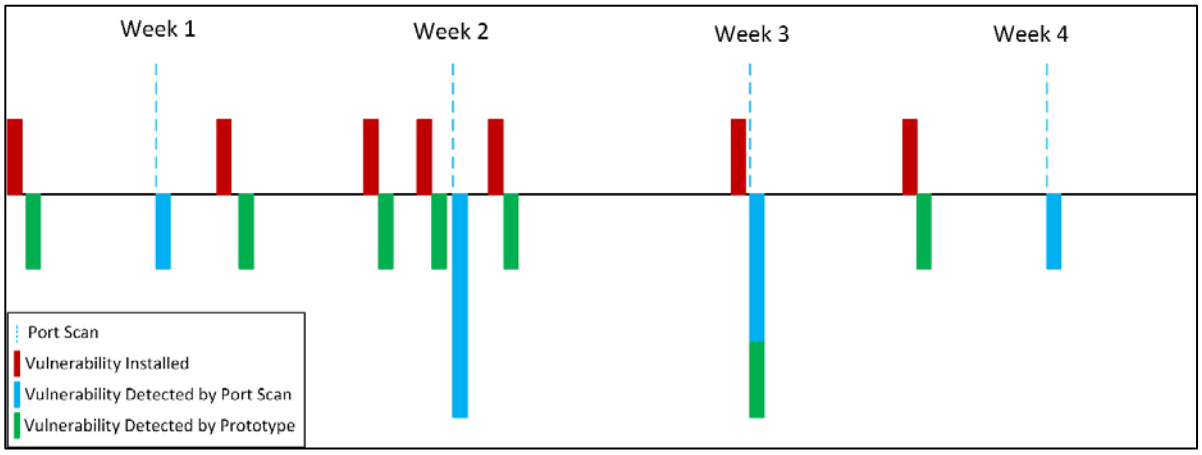

Figure 14: Speed Comparison for Weekly Port Scan

As shown in this figure, the prototype continues to detect all vulnerabilities in real-time, but because the interval between port scans has been increased to a week, vulnerable software could run on the system for up to 7 days, or 168 hours before being detected by the port scan.

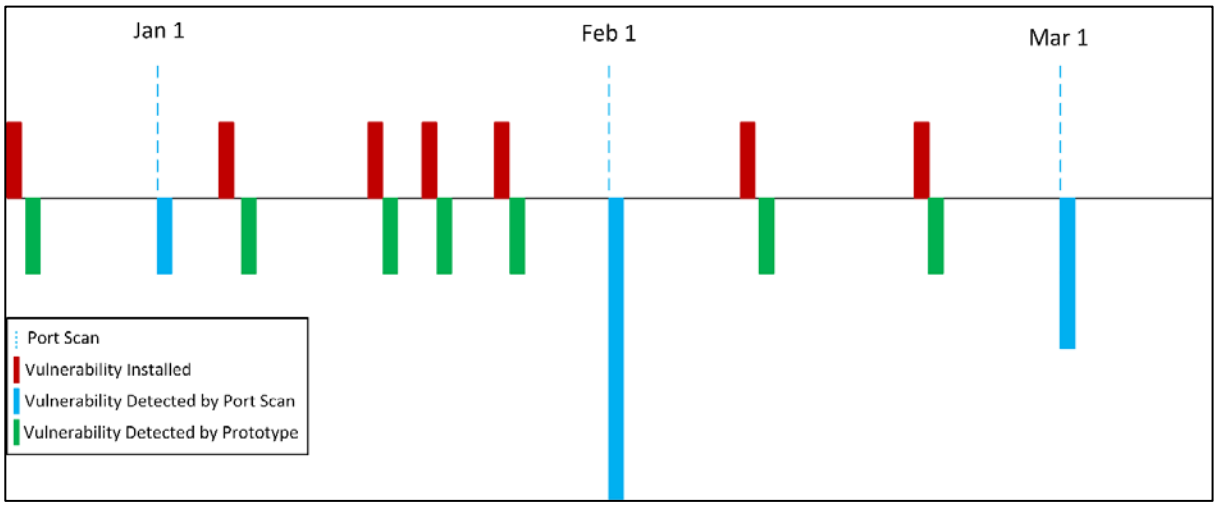

Figure 15: Speed Comparison for Monthly Port Scan

As shown in this figure, the prototype continues to detect all vulnerabilities in real-time, but because the interval between port scans has been increased to a month, vulnerable software could run on the system for up to 31 days, or 744 hours before being detected by a port scan.

As shown in Figure 13-15, when a vulnerability is installed on a machine, the prototype detects it immediately in real-time. The port scan does not detect the vulnerability until the scan is run. In Figure 13, if the vulnerability was installed at 8am, it would be detected immediately by the prototype but it would take 12 hours for the port scan to detect it. The amount of time it takes a port scan to detect a vulnerability is directly correlated to the frequency in which port scans are conducted. For example, in Figure 15, a vulnerability could be installed on January 7. This vulnerability would not be detected by a port scan for 24 days until February 1, or for a total of approximately 576 hours. These timelines show that unless a port scan is conducted immediately after vulnerable software was installed, the prototype will always be faster at reporting vulnerabilities. If the port scan is conducted immediately after the software was installed, it depends on whether the port scan or the prototype runs faster to determine which would discover the vulnerability first, though the difference would probably be a matter of seconds or minutes depending on the complexity of the port scan and the network speed, instead of a matter of days.

It is possible to compare the speed of the port scan with the time it takes the prototype to report a vulnerability from the time a log file is transmitted. However, the port scan will take longer for each client it scans in the network, and for each open port that it discovers. On average it took the Python Script about 20 seconds to run. Most of that time was spent communicating with SEA's server. The port scan took between 10 seconds and 110 seconds for

each client. This shows that the actual time that the port scan takes could be less than the prototype when there are few clients. However, in most cases, the prototype will be faster.

4.4.4 Accuracy of Discovered Vulnerabilities

To test the accuracy of the vulnerabilities reported by the prototype and the port scan, each client was manually inspected to see what versions of software are running on them compared with the results reported by the prototype and the port scan.

Table 8: Comparison of Missing Updates for Windows

	Windows 7 Manual Results	*Windows 7 ELK Results*	*Windows 7 Port Scan Results*
Missing Updates	KB2393802	KB2393802	
	KB2425227	KB2425227	
	KB2476490	KB2476490	
	KB2503658	KB2503658	
	KB2503665	KB2503665	
	KB2507938	KB2507938	
	KB2508429	KB2508429	
	KB2525694	KB2525694	
	KB2535512	KB2535512	
	KB2584146	KB2584146	
	KB2617657	KB2617657	
	KB2618451	KB2618451	
	KB2620712	KB2620712	
	KB2644615	KB2644615	
	KB2655992	KB2655992	
	KB2658846	KB2658846	
	KB2659262	KB2659262	
	KB2660649	KB2660649	
	KB2691442	KB2691442	
	KB2709715	KB2709715	
	KB2743555	KB2743555	
	KB2753842	KB2753842	
	KB2769369	KB2769369	
	KB2778930	KB2778930	
	KB2785220	KB2785220	
	KB2790113	KB2790113	
	KB2849470	KB2849470	
	KB2859537	KB2859537	
	KB2868623	KB2868623	
	KB2875783	KB2875783	
	KB2876284	KB2876284	
	KB2876331	KB2876331	
	KB2929961	KB2929961	
	KB3000061	KB3000061	
	KB3000869	KB3000869	
	KB3020393	KB3020393	

For the Windows machine, the list of installed updates was cross checked with the Microsoft Security Bulletin to look for missing updates, and then was compared with the results reported by the ELK server.

As shown in Table 8, the prototype was able to detect all of the updates that were missing from the machine, while the port scan was unable to detect a single missing update. However, the ELK server reported updates that should have been installed on a given version of Windows, but if the related software isn't present on the machine, the update may not actually be needed. For example, if the ELK server detects that the Windows machine is missing update KB2659262 for Silverlight, but Silverlight was never actually installed on the machine, then it isn't actually vulnerable, even though the update is missing. The prototype could easily allow for false positives with regards to Windows updates. Methods to negate the effects of false positives will be discussed further in Chapter 5.

For the Debian machine, the installed updates were manually compared with known vulnerabilities in the CVE database, and then was compared with the results of the prototype.

Table 9: Comparison of Discovered Vulnerabilities for Debian

	Debian 7 Manual Results	*Debian 7 Automated Results*	*Debian 7 Port Scan*
Vulnerabilities	Openssl 1.0.1c-4 Heartbleed Apache 2.2.22 DOS	Openssl 1.0.1c-4 Heartbleed Apache 2.2.22 DOS	

As shown in Table 9 the prototype was able to detect all of the vulnerabilities on the Debian machine, but the port scan failed to detect any of them.

The firmware on the Cisco switch was manually confirmed and then compared with the results of the port scan and the prototype.

Table 10: Comparison of Discovered Vulnerabilities for Cisco Switch

	Cisco Manual Results	*Cisco Automated Results*	*Cisco Port Scan*
Vulnerabilities	IOS 15.0 DDOS	IOS 15.0 DDOS	IOS 15.0 DDOS

As shown in Table 10, both the port scan and prototype were able to detect the vulnerability in the Cisco switch.

5 RESULTS

5.1 Framework Analysis

This section analyzes the final framework and answers the proposed research questions and addresses the proposed hypotheses:

- (R1) How does the log file inventory management system compare with traditional port scanning with regards to speed and accuracy?
- (R2) What are the challenges of creating a log file inventory management system and what techniques can be used to overcome them?
- (R3) What are the key indicators required from log files to cross check them with a vulnerability database?
- (H1) It is possible to generate an accurate system inventory using log files.
- (H2) It is possible to generate an accurate inventory of vulnerabilities that exist in the current configuration.
- (H3) The log file inventory management system will enable faster alerting of vulnerabilities than port scanning.

5.1.1 (R1) Comparison of Log File Prototype and Port Scanning

As shown in section 4.4 the Prototype outperformed the traditional port scanning method in all ways except for one. The prototype was more accurate than the port scanning method at

detecting the Operating Systems, installed updates and packages, and at identifying vulnerabilities. The only thing that the port scan was able to detect that the prototype didn't was the presence of OpenSSH on the Debian machine. The port scan detected it because it was running on port 22, while the prototype did not detect it because it was a package that came preinstalled before the prototype was setup to monitor for installed packages.

This is a limitation with the prototype that could be resolved with future research. One such method would be to ingest a list of pre-installed packages into ELK when the client is initially introduced to the system. Despite that one advantage that port scanning has, the results of the testing clearly indicate that the prototype is faster and more accurate than port scanning.

5.1.2 Accuracy of Software Inventory

The prototype was able to detect the correct versions of the OS and firmware, the updates, and installed packages, with a high level of accuracy on the Windows, Debian, and Cisco clients. It failed to detect anything on the printer and firewall because they were not compatible with the prototype due to their lack of identifying information in their log files. Most network devices support SNMP. Even though the log files do not contain the requisite information, the prototype could gather that information through SNMP to overcome this limitation. The port scan failed to detect the correct Operating System, any of the windows updates, and most of the installed Linux packages.

5.1.3 Speed of Methods

The prototype reported discovered vulnerabilities in real-time, usually in less than 30 seconds after the vulnerability was introduced to the system. The port scan reported discovered software whenever it was run. The software then had to be cross check for vulnerabilities.

5.1.4 Accuracy of Reported Vulnerabilities

The prototype detected all of the missing updates on Windows, the vulnerable packages on Debian, and the firmware on the Cisco Switch. The port scan didn't detect any vulnerable software. It didn't identify the missing Windows updates, or even the presence of a vulnerable version of OpenSSL on Debian. The prototype was much more accurate at reporting vulnerable software than the port scan.

5.1.5 Limitations

There are several limitations to the prototype that will be discussed in greater detail in Chapter 6. These limitations include the lack of a comprehensive vulnerability database, the problems related to keeping such a database updated, the ability to check for vulnerable software that was installed before it was known to be vulnerable, a method for reporting when vulnerabilities have been resolved, and a better method of reporting vulnerabilities that can easily show the current state of the clients.

5.2 (R2) Challenges in Creating a Log File Inventory Management System

The challenges that arose from creating a Log File Inventory Management System have been discussed in detail in section 4.3. In general, the problems that arose were related to the lack of a parsable vulnerability database, and limitations within ELK for reporting resolved vulnerabilities. The open source vulnerability databases were not normalized enough to be useful in this research. In general, the vendor databases were detailed and formatted in such a way to be conducive to this research. A more detailed analysis of a few well known vulnerability databases can be found in section 4.1.4.

5.3 (R3) Required Key Indicators from Log Files

Throughout the development of the prototype, several key indicators were identified that are required in order to be able to generate an accurate system inventory using log files. These indicators vary from vendor to vendor, but are similar in all of them. These indicators are shown in Table 11.

Table 11: Log File Key Indicators

Key Indicators	*Windows*	*Linux*	*Network Device*
Operating System	X	X	
Architecture	X	X	
Kernel		X	
Service Pack	X		
Firmware			X
Firmware Version			X
Application	X	X	
Application Version	X	X	
Windows Update	X		

Not every log file for each vendor will contain all of the key indicators, but each key indicator for a given vendor needs to be represented inside a log file so that it can be reported to the centralized server.

5.4 (H1) It is Possible to Generate an Accurate System Inventory Using Log Files

As shown from the testing results, this hypothesis was proven true if the client logs the necessary identifying information.

5.5 (H2) It is Possible to Accurately Detect Vulnerabilities Using Log Files

As shown from the testing results, this hypothesis was also proven true if the client logs the necessary identifying information.

5.6 (H3) Log Inventory Management System is Faster than Port Scanning

As shown from the testing results, this hypothesis was also proven true for almost all situations.

5.7 Research Contributions

Software vulnerabilities are not a new problem. They have existed for decades. Finding and resolving vulnerabilities is a continuous process that is difficult and time consuming. Vulnerabilities can often be overlooked until they are exploited. This research attempts to find a way to mitigate the difficulty of finding vulnerable software. Components of this research can also be used in other non-security related applications.

5.7.1 Automated Log File Inventory Management System

The framework for this research introduced a completely original method of software inventory through the use of log files. Any application that requires an automated method of inventorying the software running on network devices can benefit from this research. This framework is beneficial because it runs in real-time, is completely accurate, and can be configured to work on a wide range of computers and network devices.

5.7.2 Standard for Requisite Log Data for Vulnerability Assessment

As shown in Section 5.3, this research identified the requisite information that needs to be collected from log files to create an accurate inventory of a networked system, but also the information required to compare that information with known vulnerabilities.

5.7.3 Framework for Data Gathering and Analysis

The prototype, that was created to gather and parse log files and cross check their data with known vulnerabilities, is another significant research contribution. It shows the viability of using such an approach and can be built upon. It provides a foundation for furthering this research in the future.

6 CONCLUSIONS AND FUTURE WORK

The purpose of this research was to ascertain the viability of using log files to find and report vulnerabilities running on local systems. From this research, it was found that such an approach can be much more effective and efficient at than traditional port scanning methods at finding and reporting software vulnerabilities. A prototype was created as part of this research to demonstrate the feasibility of such an approach. There were two major components of this prototype: The centralized ELK server, and the individual clients.

6.1 Conclusions

The results of this research show high potential for this method of detecting vulnerabilities. Not, only is this method faster than tradition port scanning, but it is more accurate as well. The prototype detected installed updates and vulnerabilities with 100% accuracy. While there is a lot of work to do to turn this prototype into a reality, this research confirms that there is a very high potential for making the vulnerability assessment process easier and more accurate. This research resulted in the following key findings:

- It is possible to generate an accurate system inventory using log files
- It is possible to generate an accurate inventory of vulnerabilities that currently exist
- The prototype was faster than the port scan at reporting vulnerabilities

- The prototype was more accurate than the port scan at identifying the software running on the clients and at identifying that vulnerable software was present

While the current prototype still has some major limitations, the findings show that it has a high potential. There is very little that can be done in order to improve the port scanning method. However, there are many ways in which the prototype can be enhanced and improved.

6.2 Improvements and Recommendations

The prototype strictly used log files to gather data. In order to overcome the limitations of a log file only approach, SNMP could be added to gather the requisite data from network devices. Other technologies and techniques could also be included as deemed necessary, such as initially ingesting log files, creating customized agents to log specific data not already found in log files, and running scripts to report information such as what packages or updates have already been installed.

In order to make this prototype a reality several things need to occur. Ideally, vendors would ensure that all of their devices support the syslog protocol and log the key indicators used in this research. In cases where this does not happen, other technologies like those mentioned above could be used to supplement the log files. Due to the loose nature of the syslog protocol, someone would have to write a significant number of Logstash parsing algorithms to allow the centralized server to collect all the needed data from any number of devices. One way to do this would be to rely on crowdsourcing the technology to spread out the work among interested parties.

A complete and always up to date vulnerability database would also need to be created in order for this to work. This database would need to be properly normalized, and would need to contain a comprehensive list of vulnerabilities across all platforms and vendors. This could be created and maintained through cooperation among vendors, or perhaps through crowd sourcing as well.

6.3 Future Research

This section outlines the potential for future research and improvements including:

- Creation and maintenance of a parsable and comprehensive vulnerability database
- Create a method to keep the vulnerability database up to date.
- Automate a process to rank discovered vulnerabilities based on context, severity, and any other factors deemed relevant
- Check previously reported logs for vulnerabilities when new vulnerabilities are discovered.
- Create a better method of reporting discovered vulnerabilities that doesn't contain duplicates and shows the current state of discovered vulnerabilities
- Create and implement a methodology to report when vulnerabilities have been resolved

6.3.1 Comprehensive Vulnerability Database

In order to fully assess the potential for this prototype, a method for creating a comprehensive database needs to be developed. For this research, data was manually gathered

and compiled into a database. This process needs to be optimized, automated and expanded to cover a much bigger subset of software vulnerabilities.

6.3.2 Keep Vulnerability Database up to date

Related to creating a comprehensive database, there should be a mechanism in place to ensure that the database is always up to date, in real-time if possible. In order to catch newly discovered vulnerabilities the system has to have access to a frequently updated database.

6.3.3 Rank Discovered Vulnerabilities

When vulnerabilities are discovered, they must be prioritized in order to determine which ones need to be resolved first. Currently, the standard method for doing this is to manually rank them based on severity and the status of the vulnerable machine. Fixing a low severity vulnerability on a production machine may me more urgent than fixing a critical severity vulnerability on a sandboxed development environment. In order to fully automate the vulnerability reporting process, this component should be implemented.

6.3.4 Check for Previously Installed Vulnerabilities

If vulnerable software is installed on a machine, before a corresponding entry is made in the vulnerability database, the current prototype will not detect the vulnerability, unless the log file is sent a second time. Developing a method to check for recently discovered vulnerabilities after the software has already been installed is necessary to have a good vulnerability assessment system.

6.3.5 Better Method for Reporting

Kibana is a poor reporting tool for this framework. It is inherently time based and doesn't easily support a method for showing the current configuration of the devices on the network. It is better suited for displaying trends. Creating a custom platform to report discovered vulnerabilities and resolved vulnerabilities would greatly enhance the value and efficiency of this framework. This new reporting tool could be built on top of the existing Elasticsearch database using the Elasticsearch API.

6.3.6 Report Resolved Vulnerabilities

The centralized server needs a method to detect when vulnerabilities have been resolved and to keep resolved vulnerabilities from showing up as current vulnerabilities. This feature would make the framework a much more complete and effective vulnerability assessment tool.

REFERENCES

Apache. 2015. “Apache Vulnerabilities.” Accessed July 11. http://httpd.apache.org/security/vulnerabilities_22.html.

Bendix, L., and C. Pendleton. 2013. “The Role of Configuration Management in Outsourcing and Distributed Development.” *Proceedings of the 9th Central & Eastern ….* http://dl.acm.org.erl.lib.byu.edu/citation.cfm?id=2556610.2556615&coll=DL&dl=ACM&CFID=593073421&CFTOKEN=26469304.

Canada, Public Safety. 2014. “AL14-005: OpenSSL Heartbleed Vulnerability,” April. http://www.publicsafety.gc.ca/cnt/rsrcs/cybr-ctr/2014/al14-005-eng.aspx.

Debian. 2015. “Snapshot.debian.org.” Accessed July 11. http://snapshot.debian.org/.

Forte, D. 2005. “Log Management for Effective Incident Response.” *Network Security*. http://www.sciencedirect.com/science/article/pii/S1353485805702798.

Havens, R. 2011. “Naive Bayesian Spam Filters for Log File Analysis.” *All Theses and Dissertations*. http://scholarsarchive.byu.edu/etd/2814.

Kercher, K. 2013. “Distributed Agent Cloud-Sourced Malware Reporting Framework,” no. September. http://scholarsarchive.byu.edu/etd/4250/.

Lund, T., H. Panike, S. Moses, D. Rowe, and J. Ekstrom. 2015. “Practical Data Mining and Analysis for System Administration.” In *2015 ASEE Annual Conference and Exposition*, 26.1233.1–26.1233.24. https://peer.asee.org/practical-data-mining-and-analysis-for-system-administration.

Mell, P., K. Scarfone, and S. Romanosky. 2006. “Common Vulnerability Scoring System.” *IEEE Security and Privacy Magazine* 4 (6). IEEE: 85–89. doi:10.1109/MSP.2006.145.

NVD. 2015. “National Vulnerability Database.” Accessed January 27. http://nvd.nist.gov/.

OSVDB. 2015. “OSVDB: Open Sourced Vulnerability Database.” Accessed February 6. http://osvdb.org/.

RFC1098. 2015. “A Simple Networking Management Protocol (SNMP).” Accessed September 5. http://www.rfc-editor.org/rfc/rfc1157.txt.

Rinnan, R. 2005. “Benefits of Centralized Log File Correlation.” http://brage.bibsys.no/xmlui/handle/11250/143787.

Teo, L. 2000. “Port Scans and Ping Sweeps Explained.” *Linux Journal* 2000 (80es). Belltown Media: 2. http://dl.acm.org/ft_gateway.cfm?id=364651&type=html.

US-CERT. 2015. “US-CERT | United States Computer Emergency Readiness Team.” Accessed February 6. https://www.us-cert.gov/.

Wooldridge, M. 1998. *Agent Technology: Foundations, Applications, and Markets*. Springer Science & Business Media. http://books.google.com/books/about/Agent_technology.html?id=D5pnZ_fUplUC&pgis=1.

APPENDICES

APPENDIX A. ELK CONFIGURATION FILES

Logstash-syslog.conf

```
input {
  tcp {
    port => 5000
    type => syslog
  }
  udp {
    port => 5000
    type => syslog
  }
  udp {
    port => 514
    type => syslog
  }
  udp {
    port => 5005
    type => vulnerabilities
  }
}

filter {

    grok {
            break_on_match => false
            match => { "message" =>
"<%{POSINT:syslog_pri}>(?:%{SYSLOGTIMESTAMP:syslog_timestamp}|%{TIMESTAMP
_ISO8601:syslog_timestamp8601}) %{SYSLOGHOST:syslog_hostname} %{PROG:syslog_pr
ogram}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
            match => { "message" =>
"<%{POSINT:syslog_pri}>(?:%{SYSLOGTIMESTAMP:syslog_timestamp}|%{TIMESTAMP
_ISO8601:syslog_timestamp8601}) %{SYSLOGHOST:syslog_hostname}
(?<syslog_program>[-a-zA-Z0-
9_ ]*\[\w*\]) %{POSINT:syslog_pid} %{GREEDYDATA:syslog_message}" }

            match => { "message" =>
"\[(?<timestamp>%{DAY:day} %{MONTH:month} %{MONTHDAY} %{TIME} %{YEAR})
```

```
\] \[%{WORD:class}\] %{WORD:program}/(?<program_version>[0-
9.]*) %{GREEDYDATA:errmsg}" }

        match => { "message" => "<%{POSINT:syslog_pri}>(?<router_time>[0-
9: ]*)(?<syslog_command>[%A-Z0-9:_-]*) %{GREEDYDATA:syslog_message}" }

        match => { "message" => "%{POSINT:vuln_id} IP:%{IP:vuln_ip}
Date:%{DATE:vuln_date} Severity:%{WORD:vuln_severity}
Message:%{GREEDYDATA:vuln_message}" }

        match => { "message" => "%{USERNAME:vuln_id} IP:%{IP:vuln_ip}
Severity:%{WORD:apache_severity} Message:%{GREEDYDATA:vuln_message}" }

        match => { "message" => "%{USERNAME:vuln_id} IP:%{IP:vuln_ip}
Severity:%{WORD:cisco_severity} Type:cisco Message:%{GREEDYDATA:vuln_message}" }

        match => { "message" => "Install: %{GREEDYDATA:packages}" }
        add_field => [ "received_at", "%{@timestamp}" ]
        add_field => [ "received_from", "%{host}" ]
  }

   syslog_pri {
        type => syslog
  }
  ruby {
     code => "
        temp = event['packages']
        if temp
            array = temp.split(', ')
            program = Array.new()
            architecture = Array.new()
            version = Array.new()
            lin = 'y'
            for element in array do
                element.delete!(')')
                element.gsub!(/, automatic/, '')
                newEl = element.split(/([:])|( \()/)
                program.push(newEl[0])
                architecture.push(newEl[2])
                version.push(newEl[4])
                #event['program'][counter] = newEl[0]
                #event['architecture'][counter] = newEl[2]
                #event['version'][counter] = newEl[4]
            end
            event['program'] = program
            event['architecture'] = architecture
```

```
                    event['version'] = version
                   event['lin-pack'] = lin
               end
       "
    }
}

output {
  elasticsearch { cluster => elasticsearch
               protocol => http
 }
  stdout { codec => rubydebug }

#  stdout { debug => true }
}
```

APPENDIX B. MYSQL DATABASE CONFIGURATION

MySQL Create Table Script

```
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS
*/;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Database: `Vulnerabilities`
--

-- --------------------------------------------------------

--
-- Table structure for table `apachevulnerabilities`
--

CREATE TABLE IF NOT EXISTS `apachevulnerabilities` (
  `id` int(25) NOT NULL AUTO_INCREMENT,
  `rank` varchar(255) NOT NULL,
  `title` varchar(255) NOT NULL,
  `cve` varchar(255) NOT NULL,
  `description` varchar(500) NOT NULL,
  `affectsVersion` varchar(255) NOT NULL,
  `fixVersion` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=13 ;


-- --------------------------------------------------------

--
-- Table structure for table `ciscohosts`
--

CREATE TABLE IF NOT EXISTS `ciscohosts` (
```

```
  `ip` varchar(15) NOT NULL,
  `product` varchar(50) NOT NULL,
  `firmware` varchar(100) NOT NULL,
  `version` varchar(50) NOT NULL,
  PRIMARY KEY (`ip`,`product`,`firmware`,`version`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- --------------------------------------------------------

--
-- Table structure for table `ciscovulnerabilities`
--

CREATE TABLE IF NOT EXISTS `ciscovulnerabilities` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `rank` varchar(20) NOT NULL,
  `title` varchar(100) NOT NULL,
  `description` varchar(255) NOT NULL,
  `cve` varchar(50) NOT NULL,
  `product` varchar(50) NOT NULL,
  `affectsVersion` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;


-- --------------------------------------------------------

--
-- Table structure for table `hosts`
--

CREATE TABLE IF NOT EXISTS `hosts` (
  `ip` varchar(255) NOT NULL,
  `hostname` varchar(255) NOT NULL,
  `os` varchar(255) NOT NULL,
  `version` varchar(255) NOT NULL,
  `dt` datetime NOT NULL,
  PRIMARY KEY (`ip`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- --------------------------------------------------------

--
-- Table structure for table `linuxvulnerabilities`
```

```
--

CREATE TABLE IF NOT EXISTS `linuxvulnerabilities` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `rank` varchar(20) NOT NULL,
  `title` varchar(80) NOT NULL,
  `cve` varchar(20) NOT NULL,
  `description` varchar(250) NOT NULL,
  `affectsVersion` varchar(100) NOT NULL,
  `fixVersion` varchar(100) NOT NULL,
  `package` varchar(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;


-- --------------------------------------------------------

--
-- Table structure for table `msvulnerabilities`
--

CREATE TABLE IF NOT EXISTS `msvulnerabilities` (
  `id` int(255) NOT NULL AUTO_INCREMENT,
  `DatePosted` varchar(255) NOT NULL,
  `Bulletin` varchar(255) NOT NULL,
  `BulletinKB` varchar(255) NOT NULL,
  `Severity` varchar(255) NOT NULL,
  `Impact` varchar(255) NOT NULL,
  `Title` varchar(255) NOT NULL,
  `AffectedProduct` varchar(255) NOT NULL,
  `ComponentKB` varchar(255) NOT NULL,
  `AffectedComponent` varchar(255) NOT NULL,
  `SupercededBy` varchar(255) NOT NULL,
  `Supercedes` varchar(255) NOT NULL,
  `Reboot` varchar(255) NOT NULL,
  `CVE` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=17083 ;
```

APPENDIX C. PYTHON SCRIPTS

The following Python script was used in this research:

```
import urllib2
import requests
import base64
import json
import datetime
import pymysql
import re
import socket

db = pymysql.connect(host="localhost", # your host,
                     user="root", # your username
                      passwd="password", # your password
                      db="Vulnerabilities",) # name of the data base
cur = db.cursor()
        # you must create a Cursor object. It will let
        #  you execute all the queries you need

#Query ELK for Windows Updates
url = 'http://localhost:9200/_search?q=syslog_pid:19&size=300&sort=@timestamp:desc'
req = urllib2.Request(url)
out = urllib2.urlopen(req)
data = out.read()

data = json.loads(data)

total = data['hits']['total']
for each in range (0, total):
        bit = "none"
        message = data['hits']['hits'][each]['_source']['syslog_message']

        results = re.search('(KB[0-9]*)', message)
        if results is not None:
                if len(results.group(1)) != 0:
                        update = results.group(1)
```

```
        results = re.search('(x64)', message)
        if results is not None:
                if len(results.group(1)) != 0:
                        bit = results.group(1)

        results = re.search('(32-bit)', message)
        if results is not None:
                if len(results.group(1)) != 0:
                        bit = results.group(1)

        ip = data['hits']['hits'][each]['_source']['received_from']
        hostname = data['hits']['hits'][each]['_source']['syslog_hostname']
        timestamp = data['hits']['hits'][each]['_source']['received_at']

# Insert a list of installed Windows Updates to track them.
        query = "INSERT IGNORE INTO updates (ip, hostname, updates, dt) VALUES ('" + ip
+ "', '" + hostname + "', '" + update + "', '" + timestamp + "');"
        cur.execute(query)
        db.commit()


#Query ELK for Windows OS and SP

url = 'http://localhost:9200/_search?q=syslog_pid:6009&size=1&sort=@timestamp:desc'
req = urllib2.Request(url)
out = urllib2.urlopen(req)
data = out.read()

data = json.loads(data)

windows = data['hits']['hits'][0]['_source']['syslog_message'].split()

version = windows[4]
if version == "6.01.":
        version = "7"
windowsOS = windows[2] + " " + version
windowsSP = "Service Pack " + windows[8][0]
ip = data['hits']['hits'][0]['_source']['received_from']
hostname = data['hits']['hits'][0]['_source']['syslog_hostname']


#Check for missing Windows Updates
query = "SELECT DISTINCT(ComponentKB), DatePosted, Severity, Impact, Title,
AffectedComponent, SupercededBy, Supercedes, CVE, BulletinKB FROM msvulnerabilities
WHERE AffectedProduct LIKE '%" + windowsOS + "%' AND AffectedProduct LIKE  '%" + bit
```

```
+ "%'  AND AffectedProduct LIKE '%" + windowsSP + "%' AND SupercededBy = '' AND
AffectedComponent = '' GROUP BY ComponentKB ORDER BY ComponentKB desc";

cur.execute(query)
issues = []
for row in cur.fetchall():
        issues.append(row)
entries = []
toRemove = []
for entry in issues:
        query = "SELECT * FROM updates WHERE (updates = 'KB" + entry[0] + "' OR updates
= 'KB" + entry[9] + "') AND ip = '" + ip + "';"
        cur.execute(query)
        #print query
        if cur.rowcount == 0:
                entries.append(entry)

update = ""
for entry in entries:
        for entry2 in entries:
                val = entry[7]
                kb = entry2[0]
                results = re.search('MS[0-9]*-[0-9]*\[([0-9]*)\]', val)
                if results is not None:
                        if len(results.group(1)) != 0:
                                update = results.group(1)
                if update != "":
                        if kb == update:
                                toRemove.append(entry2)
                                #entries.remove(entry2)
                                update = ""
print len(toRemove)
toRemove = list(set(toRemove))
print len(toRemove)
for entry in toRemove:
        entries.remove(entry)
#Send discovered missing updates back to ELK
for entry in entries:

        UDP_IP = "127.0.0.1"
        UDP_PORT = 5005
        MESSAGE = entry[0] + " IP:" + ip + " Date:" + entry[1] + " Severity:" + entry[2] + "
Message:" + entry[4]
        MESSAGE = entry [0]

        sock = socket.socket(socket.AF_INET, # Internet
```

```
                socket.SOCK_DGRAM) # UDP
        sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))



# Query ELK for Apache vulnerabilities
url = 'http://localhost:9200/_search?q=program:Apache&size=1&sort=@timestamp:desc'


req = urllib2.Request(url)
out = urllib2.urlopen(req)
data = out.read()

# returned data is JSON
data = json.loads(data)
num = data['hits']['total']

message = data['hits']['hits'][0]['_source']['program_version']
ip = data['hits']['hits'][0]['_source']['host']
date = data['hits']['hits'][0]['_source']['timestamp']


query = "SELECT title, rank, cve FROM apachevulnerabilities WHERE affectsVersion LIKE
'%" + message + "%'"

cur.execute(query)
#Send Apache vulnerabilities back to ELK
for row in cur.fetchall():

        UDP_IP = "127.0.0.1"
        UDP_PORT = 5005
        MESSAGE = row[2] + " IP:" + ip + " Severity:" + row[1] + " Message:" + row[0]

        print "message:", MESSAGE

        sock = socket.socket(socket.AF_INET, # Internet
                socket.SOCK_DGRAM) # UDP
        sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))

# Query ELK for Linux Distribution Info
url = 'http://localhost:9200/_search?q=syslog_pri:5&size=300&sort=@timestamp:desc'

req = urllib2.Request(url)
out = urllib2.urlopen(req)
data = out.read()
```

```
data = json.loads(data)
num = data['hits']['total']

for each in range (0, num):
        message = data['hits']['hits'][each]['_source']['syslog_message']
        results = re.search('Linux version ([amd0-9.-]*)', message)
        if results is not None:
                if len(results.group(1)) != 0:
                        update = results.group(1)
                        print update

#Query ELK for installed Linux Packages
url = 'http://localhost:9200/_search?q=lin-pack:y'
req = urllib2.Request(url)
out = urllib2.urlopen(req)
data = out.read()
data = json.loads(data)
ip = data['hits']['hits'][0]['_source']['received_from']
num = data['hits']['total']

for each in range (0, num):
        programs = data['hits']['hits'][each]['_source']['program']
        versions = data['hits']['hits'][each]['_source']['version']

        length = len(programs)
        for i in range (0, length):
                query = "INSERT IGNORE INTO packages (ip, hostname, package, version)
VALUES ('" + ip + "', '" + ip + "' , '" + programs[i] + "', '" + versions[i] + "');"
                cur.execute(query)
                db.commit()

        for i in range (0, length):
                query = "SELECT * FROM linuxvulnerabilities WHERE package='" +
programs[i] + "' AND affectsVersion='" + versions[i] + "';"
                cur.execute(query)

# Send Linux Vulnerabilities back into ELK
                for row in cur.fetchall():
                        UDP_IP = "127.0.0.1"
                        UDP_PORT = 5005
                        MESSAGE = row[3] + " IP:" + ip + " Severity:" + row[1] + " Message:" +
row[2]

                        print "message:", MESSAGE

                        sock = socket.socket(socket.AF_INET, # Internet
```

```
socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))

# Query SEA for Cisco Switches
url =
'https://sea.byu.edu/es/_search?q=description:C3560%20Software&analyze_wildcard=true&size
=1000'

req = urllib2.Request(url)
base64string = "**************" #base64 encoded password
print base64string
req.add_header("Authorization", "Basic %s" % base64string)


out = urllib2.urlopen(req)
data = out.read()

data = json.loads(data)
num = data['hits']['total']

for each in range (0, 99):
        description = data['hits']['hits'][each]['_source']['description']
        ip = data['hits']['hits'][each]['_source']['logSourceIP']
        firmware = ""
        version = ""
        results = re.search('Cisco IOS Software, C3560 Software \(([A-z0-9- ]*)\)', description)
        if results is not None:
                if len(results.group(1)) != 0:
                        update = results.group(1)
                        firmware = update

        results = re.search('Cisco IOS Software, C3560 Software \([A-z0-9- ]*\), Version ([0-
9\.]*)', description)
        if results is not None:
                if len(results.group(1)) != 0:
                        update = results.group(1)
                        version = update

        query = "INSERT IGNORE INTO ciscohosts (ip, product, firmware, version) VALUES
('" + ip + "', 'Cisco Catalyst 3560', '" + firmware + "', '" + version + "');"

        cur.execute(query)
        db.commit()
```

```
        query = "SELECT * FROM ciscovulnerabilities WHERE product='Cisco Catalyst 3560'
AND affectsVersion='" + version + "';"
        cur.execute(query)

# Send Cisco Vulnerabilities into ELK
        for row in cur.fetchall():

                UDP_IP = "127.0.0.1"
                UDP_PORT = 5005
                MESSAGE = row[4] + " IP:" + ip + " Severity:" + row[1] + " Type:cisco" + "
Message:" + row[2]

                sock = socket.socket(socket.AF_INET, # Internet
                        socket.SOCK_DGRAM) # UDP
                sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

APPENDIX D. COMPARISON FOR WINDOWS 7

	Windows 7 Manual Results	*Windows 7 Prototype Results*	*Windows 7 Port Scan*
IP Address	192.168.230.137	192.168.230.137	192.168.230.137
OS	Windows 7	Windows 7	Windows Server 2008 R2
Architecture	64 Bit	64 Bit	
Installed Updates	KB2461484	KB2461484	
	KB2479943	KB2479943	
	KB2491683	KB2491683	
	KB2506014	KB2506014	
	KB2506212	KB2506212	
	KB2506928	KB2506928	
	KB2509553	KB2509553	
	KB2511455	KB2511455	
	KB2515325	KB2515325	
	KB2532531	KB2532531	
	KB2533552	KB2533552	
	KB2533623	KB2533623	
	KB2536275	KB2536275	
	KB2536276	KB2536276	
	KB2544893	KB2544893	
	KB2545698	KB2545698	
	KB2547666	KB2547666	
	KB2552343	KB2552343	
	KB2560656	KB2560656	
	KB2563227	KB2563227	
	KB2564958	KB2564958	
	KB2570947	KB2570947	
	KB2574819	KB2574819	
	KB2579686	KB2579686	
	KB2585542	KB2585542	
	KB2592687	KB2592687	
	KB2603229	KB2603229	
	KB2604115	KB2604115	
	KB2619339	KB2619339	
	KB2620704	KB2620704	
	KB2621440	KB2621440	
	KB2631813	KB2631813	
	KB2639308	KB2639308	
	KB2640148	KB2640148	
	KB2647753	KB2647753	
	KB2653956	KB2653956	
	KB2654428	KB2654428	

	KB2660075	KB2660075
	KB2667402	KB2667402
	KB2670838	KB2670838
	KB2676562	KB2676562
	KB2685811	KB2685811
	KB2685813	KB2685813
	KB2685939	KB2685939
	KB2690533	KB2690533
	KB2698365	KB2698365
	KB2705219	KB2705219
	KB2706045	KB2706045
	KB2712808	KB2712808
	KB2718704	KB2718704
	KB2719857	KB2719857
	KB2726535	KB2726535
	KB2727528	KB2727528
	KB2729094	KB2729094
	KB2729452	KB2729452
	KB2731771	KB2731771
	KB2732059	KB2732059
	KB2732487	KB2732487
	KB2736422	KB2736422
	KB2742599	KB2742599
	KB2750841	KB2750841
	KB2758857	KB2758857
	KB2761217	KB2761217
	KB2763523	KB2763523
	KB2770660	KB2770660
	KB2773072	KB2773072
	KB2786081	KB2786081
	KB2789645	KB2789645
	KB2791765	KB2791765
	KB2798162	KB2798162
	KB2799926	KB2799926
	KB2800095	KB2800095
	KB2803821	KB2803821
	KB2807986	KB2807986
	KB2808679	KB2808679
	KB2813347	KB2813347
	KB2813430	KB2813430
	KB2820331	KB2820331
	KB2830477	KB2830477
	KB2832414	KB2832414
	KB2834140	KB2834140
	KB2836942	KB2836942
	KB2836943	KB2836943
	KB2839894	KB2839894
	KB2840149	KB2840149
	KB2840631	KB2840631
	KB2841134	KB2841134
	KB2843630	KB2843630
	KB2846960	KB2846960
	KB2847077	KB2847077
	KB2847311	KB2847311
	KB2847927	KB2847927
	KB2849696	KB2849696

	KB2849697	KB2849697
	KB2852386	KB2852386
	KB2853952	KB2853952
	KB2855844	KB2855844
	KB2857650	KB2857650
	KB2861191	KB2861191
	KB2861698	KB2861698
	KB2862152	KB2862152
	KB2862330	KB2862330
	KB2862335	KB2862335
	KB2862966	KB2862966
	KB2862973	KB2862973
	KB2864058	KB2864058
	KB2864202	KB2864202
	KB2868038	KB2868038
	KB2868116	KB2868116
	KB2868626	KB2868626
	KB2871997	KB2871997
	KB2872339	KB2872339
	KB2882822	KB2882822
	KB2884256	KB2884256
	KB2887069	KB2887069
	KB2888049	KB2888049
	KB2891804	KB2891804
	KB2892074	KB2892074
	KB2893294	KB2893294
	KB2893519	KB2893519
	KB2894844	KB2894844
	KB2900986	KB2900986
	KB2908783	KB2908783
	KB2911501	KB2911501
	KB2912390	KB2912390
	KB2913152	KB2913152
	KB2918077	KB2918077
	KB2918614	KB2918614
	KB2919469	KB2919469
	KB2922229	KB2922229
	KB2923545	KB2923545
	KB2926765	KB2926765
	KB2928562	KB2928562
	KB2929733	KB2929733
	KB2931356	KB2931356
	KB2937610	KB2937610
	KB2939576	KB2939576
	KB2943357	KB2943357
	KB2952664	KB2952664
	KB2957189	KB2957189
	KB2957503	KB2957503
	KB2957509	KB2957509
	KB2961072	KB2961072
	KB2965788	KB2965788
	KB2966583	KB2966583
	KB2968294	KB2968294
	KB2970228	KB2970228
	KB2971850	KB2971850
	KB2972100	KB2972100

	KB2972211	KB2972211
	KB2972280	KB2972280
	KB2973112	KB2973112
	KB2973201	KB2973201
	KB2973351	KB2973351
	KB2976627	KB2976627
	KB2976897	KB2976897
	KB2977292	KB2977292
	KB2977728	KB2977728
	KB2978092	KB2978092
	KB2978120	KB2978120
	KB2978668	KB2978668
	KB2978742	KB2978742
	KB2979570	KB2979570
	KB2980245	KB2980245
	KB2984972	KB2984972
	KB2984976	KB2984976
	KB2984981	KB2984981
	KB2985461	KB2985461
	KB2991963	KB2991963
	KB2992611	KB2992611
	KB2993651	KB2993651
	KB2993958	KB2993958
	KB2994023	KB2994023
	KB3000483	KB3000483
	KB3001554	KB3001554
	KB3002885	KB3002885
	KB3003057	KB3003057
	KB3003743	KB3003743
	KB3004361	KB3004361
	KB3004375	KB3004375
	KB3004394	KB3004394
	KB3005607	KB3005607
	KB3006121	KB3006121
	KB3006226	KB3006226
	KB3006625	KB3006625
	KB3008627	KB3008627
	KB3008923	KB3008923
	KB3009736	KB3009736
	KB3010788	KB3010788
	KB3011780	KB3011780
	KB3012176	KB3012176
	KB3013126	KB3013126
	KB3013410	KB3013410
	KB3013455	KB3013455
	KB3014406	KB3014406
	KB3019215	KB3019215
	KB3019978	KB3019978
	KB3020338	KB3020338
	KB3020388	KB3020388
	KB3021674	KB3021674
	KB3021952	KB3021952
	KB3022777	KB3022777
	KB3023266	KB3023266
	KB3023562	KB3023562
	KB3023607	KB3023607

	KB3025390	KB3025390
	KB3029944	KB3029944
	KB3031432	KB3031432
	KB3034196	KB3034196
	KB915597	KB915597
	KB971033	KB971033
	KB976902	KB976902
	KB982018	KB982018