PREDICTING VULNERABILITY RISKS USING SOFTWARE CHARACTERISTICS

A dissertation submitted to:

Kent State University Graduate School of Management

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

by

Yaman Roumani

March, 2012

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Dissertation written by

Yaman Roumani

B.S., Indiana University of Pennsylvania, 2005

M.B.A., Indiana University of Pennsylvania, 2006

M.S., Temple University, 2008

Ph.D., Kent State University, 2012

Approved by:

_____ Chair, Doctoral Dissertation Committee

Dr. Alan Brandyberry

_____ Members, Doctoral Dissertation Committee

Dr. Murali Shanker

_____

Dr. B. Eddy Patuwo

_____

Dr. Tuo Wang

Accepted by:

_____ Doctoral Director, Graduate School of Management

Dr. Murali Shanker

_____ Dean, Graduate School of Management

Dr. Kathryn S. Wilson

Acknowledgements

I am grateful to Dr. Alan Brandyberry, my dissertation advisor, for his support and mentorship throughout both my dissertation and graduate education. I would also like to thank my dissertation committee members, Dr. Murali Shanker, Dr. B. Eddy Patuwo and Dr. Tuo Wang for their helpful feedback on this study. I am also grateful to my wife, my parents, and my brother (Yaz) for their continuous support and encouragement.

TABLE OF CONTENTS

CHAPTER 1 Introduction

1.1 Introduction

Today, businesses and individuals are becoming increasingly dependent on computer systems, software applications and the Internet. With this increased reliance often comes a sense of the importance of maintaining security and guarding information over such systems. Additionally, significant costs are incurred by software vendors and clients to develop, maintain and invest in software products. Therefore, it's essential for both parties to secure and protect their investment against any potential threats which can put their investment at risk (Allen 2007).

Software vulnerabilities have been regarded as one of the key reasons for computer security breaches which resulted in billions of dollars in losses per year (Telang and Wattal 2005). With the growth of the software industry and the Internet, the number of vulnerability attacks and the ease with which an attack can be made have increased. Furthermore, since software developers are not always aware of security breaches in their code, vulnerabilities will inevitably continue to be discovered after the software is released. If there is no guarantee to deliver vulnerability-free software, it may be mandatory to improve vulnerability prediction and prevention measures.

During code development process, developers are often faced with complex programs consisting of thousands, even millions lines of code. Such complexity accompanied by limited resources and time constrains can lead to potential logical and syntactical errors which lead to vulnerabilities. Those vulnerabilities can slip by developers and software testers undetected and make it to the final product (Geer 2007, McGraw 2006, Hoglund and McGraw 2004). Hence,

software vendors have to deal with the challenges of monitoring their products and releasing patches to protect their clients against such vulnerabilities. From a software developer's perspective, releasing a perfectly secure product is not feasible and vulnerabilities will inevitably be discovered after the software is released to the market.

Oftentimes, software vendors depend on different means to secure their software during and after the release. Software vendors use existing developers and/or hire testers and security firms to monitor software security, fix vulnerabilities and release patches in case of any security threat. The downside of this approach is that it's not predictive, meaning that software developers have to either search for and discover vulnerabilities themselves or they have to respond to outside reports and disclosures from third parties such as clients, auditors and white/black hat hackers. Furthermore, this approach requires continuality which can be very demanding and costly for the software vendor. In practice, software testers are taught to think through the selection of input scenarios that are likely to force the software to fail and to consider symptoms of software failure. Indeed, in a perfect world with perfect specifications and a flawless understanding of security implications, finding and fixing traditional bugs/flaws would eliminate all vulnerabilities. But, unfortunately, vulnerabilities do not fit the standard description of software bugs/flaws and the symptoms of vulnerabilities are very different from those of traditional bugs. Therefore, relating vulnerabilities to traditional bugs is considered ineffective for software security.

It's my belief that by having a predictive model which forecasts vulnerabilities, software vendors can utilize their resources more efficiently and carefully plan preventive measures against such

security threats. Additionally, a vulnerability predictive model would help software developers in identifying the most vulnerable part of the software which is more likely to be exposed.

1.2 Defining Vulnerability

According to Arbaugh et al. (2000), a vulnerability is a "technological flaw in an information technology product that has security or survivability implications". Further studies defined software vulnerability as "a weakness in a system that can be exploited to violate the system's intended behavior (De Ru and Eloff 1996)" and "a bug, flaw, behavior, output, outcome or event within an application, system, device, or service that could lead to an implicit or explicit failure of confidentiality, integrity, or availability" (Schiffman 2007). According to Krusl (1998), the definition of software vulnerability in literature takes on three different forms: access control vulnerability, state space vulnerability, and fuzzy vulnerability. An access control vulnerability refers to any action which causes a system to perform operations in conflict with the security policy as defined by the access control matrices. A state space vulnerability on the other hand is a characterization of a vulnerable state in the system which distinguishes it from all non-vulnerable states. A fuzzy vulnerability refers to a violation of the expectations of users, administrators, and designers. In his work, Krusl (1998) unified those definitions into a single one which referred to a vulnerability as "an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy."

Usually, software vulnerabilities are the result of logical programming issues as well as misconfiguration in the software or hardware which may have an impact on overall system

security. For the remainder of this paper, we will use Krusl's definition of vulnerability and we will refer to software vulnerability as a security problem which can lead to violation of confidentiality, integrity or availability (CIA) of a system as a result of software bugs or design flaws.

1.3 Vulnerability Prediction Metrics

In literature, there exists two frequently used vulnerability prediction metrics: prediction at the code level and prediction at the system level. At the code level, a count of the number of bugs/flaws is used to forecast the number of vulnerabilities, whereas at the system level, a count of existing vulnerabilities is used to predict future vulnerability trends. Predicting vulnerabilities at the code level requires having access to the source code of the system or application. Also, at this level, the analysis of bugs/flaws is done at a very low level of abstraction. Using this approach has some limitations. First, having direct access to the source code depends on the development type of the system or software. This implies that open source products would be favored for analysis over closed source (proprietary) products. Also, due to code complexity, there is a possibility of missing some bugs/flaws and having some false positives (Manadhata and Wing 2005). Predicting vulnerabilities at the system level implies counting the number of times a system has been vulnerable and then building a prediction model based on the data. The main limitation of this metric is that it does not take into consideration the design and characteristics of the system which gave rise to the vulnerability in the first place (Manadhata and Wing 2005). Finally, both of these metrics produce insufficient information regarding the vulnerabilities as they only predict a count of vulnerabilities and in some cases a timeline.

Few studies have considered vulnerability prediction metrics at a third level which lies somewhere between code and system levels (Parrend and Frenot 2008). This level deals with component-based software products which consist of multiple components that are linked to each other through shared classes and objects. This strong linkage between components implies dependencies among them (Abate et al. 2009). Predicting vulnerabilities at this level is done very similar to prediction at the code level. First, the source code is scanned for bugs/flaws. Second, each vulnerability is identified and linked to its parent component; but the major problem here is correctly linking vulnerabilities to their components. The strong dependencies among components prevent us from precisely pinpointing the vulnerable components which can result in false positives (Parrend and Frenot 2008). Another limitation of this metric is that it relies on having a direct access to software code which makes the analysis limited only to open source products. This implies that closed source software will not be considered.

The approach undertaken in this research will overcome the limitations mentioned in those metrics. The approach will rely on vulnerability prediction at the system level while considering system characteristics in the analysis. Given that vulnerability prediction at system level does not require access to software code or components, this study will consider both open and closed source software products. Furthermore, we will extend the results further to include more in depth information regarding vulnerability risks.

1.4 Contribution

The majority of studies concerning software vulnerabilities are dedicated to vulnerability detection and prevention and a few devoted to vulnerability predictions and forecasting.

Furthermore, there has been a lack of quantitative studies in this area as most vulnerability studies rely on qualitative methods. In their paper, Alhazmi and Malaiya (2006) called for the need for more vulnerability predictive studies which can be evaluated quantitatively. Despite the existence of few predictive studies (Rescorla 2004, Alhazmi and Malaiya 2005a 2005b 2006, Gopalakrishna and Spafford 2005, and Woo et al. 2006), those studies relied on probabilistic models called vulnerability discovery models (VDMs). VDMs analyze historical data of vulnerabilities to predict future vulnerabilities. Although VDMs have been used in literature, such models have failed to satisfy four assumptions about their data, namely, time & effort, operational environment, independence, and static code. In his paper, Ozment (2007) thoroughly discussed the problem with each assumption. The first assumption states that quantifying time and effort spent in discovering a vulnerability reflect the accuracy of the data. Unfortunately, due to the nature of vulnerability data, such information is not easily obtainable; therefore VDMs do not consider them. The second assumption requires the environment from which a vulnerability was discovered to be similar to the post vulnerability-discovery environment. VDM fail to satisfy this assumption since vulnerabilities are often discovered through abnormal change of the environment (ex: using external script to expose a vulnerability). The third assumption states that vulnerability discovery occur independently. However, when a new class of vulnerability is discovered (ex: SQL Injections), this often leads to the discovery of other types of similar vulnerabilities. Finally, VDMs assume that software run static code. However, software code is regularly changed through updates and security patches. Based on these limitations, it seems that the reality of how vulnerabilities are discovered does not meet the assumptions of VDMs. Therefore, the validity of such models is questioned (Ozment 2007). Furthermore, existing

VDMs have relied on the cumulative number of vulnerabilities as the only dependent variable without taking into consideration other variables. For instance, vulnerability severity, frequency and type are among the dependent variables which can be used in a predictive model.

The research undertaken in this study will take on a new dimension and propose a novel vulnerability prediction model using software characteristics. This study will show that software characteristics can reveal patterns and trends that will offer new insights into the nature of vulnerability prediction. Additionally, this model will overcome some of the drawbacks and weakness of VDMs by considering a set of new measures of vulnerabilities and by relying on multiple dependent variables as the basis of measure for vulnerability discovery. The prediction capabilities of the proposed model will be examined to predict the vulnerability severity, vulnerability frequency and diversity of vulnerability types. Such prediction capabilities will expand the understanding of vulnerability prediction beyond the existing literature. The applicability of the proposed model will be analyzed using existing dataset of vulnerabilities. In addition to developing a vulnerability prediction model, the contribution of this study will be centered towards generalizing the proposed model over different categories of systems and software applications. The data analysis of this study will add to the understanding of vulnerability prediction models and will suggest directions for future research in this area.

In addition to the data analysis, a survey questionnaire will be conducted to target IT practitioners. The survey will be designed to gain an insight on how IT practitioners predict vulnerability risks using software characteristics. Based on the data analysis and survey results, I will be able to make some important research and practical contributions. First, the results will

gain insight into how IT practitioners predict vulnerability risks using software characteristics in the real world. Second, the results are foreseen to help software companies and IT practitioners in analyzing their own vulnerability risks assessment and implementing any needed changes which can increase their capabilities in providing more secure software products. Third, the results will focus on finding a link between research and practical viewpoints and providing a more comprehensive understanding of the model and how it can be applied in the real world. It's my belief that the predictive capabilities obtained from the proposed model combined with the survey results will provide researchers and practitioners with a more in depth understanding of vulnerability risks and how software characteristics can be used as predictive measures.

CHAPTER 2 Literature Review, Research Model and Hypotheses

2.1 Literature Review

Information security is considered a crucial issue for organizations in different disciplines including telecommunications, banking, retailing…etc. Recent security surveys by Luftman and McClean (2005) showed that most organizations consider information security as highly important to achieve their overall objectives. Moreover, a survey released by Ernst and Young (2008) showed that almost half of the fastest growing companies in the US have experienced an information security breach. With the discovery of each security breach, the growing awareness and importance of information security continues to increase. Research studies in this field have spanned across numerous areas such as web applications, system software and mobile devices and covered various topics including intrusion detection (Gomez and Dasgupta 2002), privacy assurance (Stathopoulos et al. 2008) and vulnerability analysis (Eusgeld et al. 2009). In software security literature, there has been little research on vulnerability evaluation methods and predictive models. Earlier studies in this field have focused on measuring vulnerabilities and vulnerable systems. One of the first evaluation methods of security and vulnerabilities was done by Littlewood et al. (1993). In their paper, Littlewood et al. (1993) measured security of a system from an operational security perspective. By analyzing the similarities between reliability and security, the authors found that the use of effort of the attacker would be a suitable measure of vulnerabilities and software security. A quantitative intruders' effort model was later constructed by Jonsson and Olovsson (1997) using real data. Alves-Foss & Barbosa (1995) were among the

earliest studies which introduced a security index to measure vulnerabilities. In their paper, the authors proposed Security Vulnerability Index (SVI) to assess the vulnerability of computer systems to common intrusion methods. The main objective of the SVI mechanism was to create a model which can be used as a standard for measuring vulnerabilities. This mechanism used three factors to calculate SVI value, namely system characteristics, neglectful acts and malevolent acts. Based on defined sets of SVI rules, the SVI value would reveal system vulnerabilities. One of the major drawbacks of SVI was the lack of measure of system factors. The authors provided a formula for calculating the SVI value, but they did not provide specific guidelines on how to retrieve system factors. Additionally, the SVI mechanism was designed to focus on operating systems and not individual software applications.

The next wave of vulnerability studies was focused towards predicting vulnerabilities using different methods such as attack trees, privilege graphs and vulnerability density functions. Dacier et al. (1996) and Ortaelo et al. (1999) proposed to model vulnerabilities of a UNIX system as a privilege graph where every node represents user privileges and every edge represents a vulnerability. The model can be used to construct a variety of ways where the attacker can exploit vulnerabilities and gain user privileges. The privilege graph is then transformed to a Markov chain based on successful attack patterns. Furthermore, Browne et al. (2001) used vulnerability data of Computer Emergency Readiness Team (CERT) to analyze vulnerability incidents trends and concluded that the cumulative number of vulnerability incidents which are reported to CERT is related to square root of time. Their proposed mathematical model predicts severity of future vulnerability exploitations based on earlier vulnerability reports. Additionally, Shin and William (2008) performed statistical analysis on

nine complexity metrics and showed that complexity metrics can be used to predict vulnerabilities. But since their analysis was performed on JavaScript Engine of Mozilla application framework, they concluded that their results might not be generalized to other projects. The latest vulnerability predictive studies managed to model vulnerabilities using density analogies. Vulnerability density relies on the number of vulnerabilities found per x lines of code to predict the future number of undiscovered vulnerabilities based on the maturity of the software. Vulnerability density assumes that different software versions are developed in a similar manner and are comparable to each other and have static code. This type of predictive study requires the use of vulnerability discovery models (VDM) which is described in the next section.

2.2 Vulnerability Discovery Models

According to Ozment (2007), vulnerability discovery models (VDM) are probabilistic models used to specify the dependency of the vulnerability discovery process on the factors that affect it. VDMs are based on software reliability models (SRM); therefore their assumptions about the data are often the same for both models. Although VDMs have been used in literature, some of those models have shortcomings regarding their assumptions (Ozment 2007). It was concluded that researchers should clearly state the assumptions upon which their models rely and define the terms that they use (Ozment 2007).

Existing VDMs utilize related models from other fields and apply them to vulnerability discovery. Using a thermodynamics analogy, Anderson (2002) proposed the Anderson Thermodynamic (AT) model for vulnerability-finding rate. The AT model argues that the

number of vulnerabilities discovered is inversely proportional to time. The applicability of the AT model was limited as it was only tested by Alhazmi and Malaiya (2008). It was found that the model wasn't a good fit for major operating systems vulnerability data.

The majority of VDMs are time-based models which maintain the total number of vulnerabilities by calendar time and consider calendar time as the independent variable (Woo 2006). Among the existing VDMs is the work by Gopalakrishna and Spafford (2005). Based on a defined set of software criteria (i.e. software which is not an operating system and is widely deployed for at least two years with significant vulnerabilities), the authors analyzed vulnerability data of IIS, BIND, Lpd, Sendmail and RPC to observe trends in data and determine if existing vulnerabilities suggest new information. The authors concluded that the discovery of vulnerability in a software artifact may influence the discovery of more vulnerabilities of the same type in that artifact. They found that measuring vulnerability occurrences can be predictive but claimed that the results may not be applicable to every other software artifact. Along their side, Alhazmi and Malaiya (2005a) proposed two VDMs: Alhazmi-Malaiya logistic model (AML) and Alhazmi-Malaiya effort model (AME). AML is an S-shaped, time-based model which considers calendar time as the independent variable and assumes that vulnerability discovery occurs in three consecutive phases. The first phase is called a learning phase where few vulnerabilities emerge while system attackers and testers learn about the targeted system. The second phase, named, linear phase, starts as the user base increases and the targeted system becomes more popular. The linear phase is considered the most critical phase since attackers become more accustomed and understanding of the system and the number of vulnerabilities grows linearly. The last phase, namely, the saturation phase, starts when a new version is released and users start switching. At this phase,

the number of vulnerabilities decreases as attackers move to the new version. The AME model on the other hand is an effort-based model which approximates effort with the number of system users (i.e. number of installations). The effort-based model takes into consideration the lifetime changes which occur in the environment of the targeted system. AME considers changes in the number of installations as the main environmental change. Since it's much more rewarding to discover vulnerabilities in systems with a large user base, therefore it's anticipated that a larger effort of vulnerability discovery would be directed toward systems with larger user base (Woo 2006).

Both AML and AME models have been tested for goodness-of-fit by numerous studies. Based on vulnerability data of Windows 98 and NT, Alhazmi and Malaiya (2005a) found that both AML and AME models provide a good fit with AME outperforming AME model. Similarly, Alhazmi and Malaiya (2005b) compared the fit of their models based on vulnerability data of Windows 95, XP, and Linux (RedHat version 6.2) operating systems and found that the AML model provides the best overall fit. Moreover, Woo et al. (2006) fitted the proposed model of Alhazmi and Malaiya (2005a) to three major web-browsers, namely, Internet Explorer, Mozilla and Firefox. They concluded that the data fit the model well even when the vulnerabilities are categorized by severity and vulnerability type. Finally, Kim et al. (2007) were among the first studies to extend AML in order to account for successive versions of software rather than considering a specific version. In their study, the authors created Multi-version Software Vulnerability Discovery Model (MVDM) and fitted the model to vulnerability data obtained from various versions of Apache web server and MySQL database. They studied the relationship between shared vulnerabilities and code size among software versions and concluded that

vulnerabilities are discovered for older software versions because of code sharing with newer version. Other than goodness-of-fit studies, the AML model was tested for predictive accuracy based on vulnerability data of Windows 98, 2000, and Linux (RedHat 7.1) (Alhazmi and Malaiya 2006). In their paper, the authors built a constrained model (AML-C) based on AML with vulnerability density information obtained from previous versions of the software. The results concluded that the model has good predictive capabilities when constraints are used.

2.3 Vulnerability Risks Prediction Model

In literature, software characteristics are classified into internal and external ones (Krusl 1998). Internal characteristics such as size, complexity and coupling are of interest mostly to software developers and engineers as they are related to the development of software. External characteristics on the other hand are properties/features which are relevant to final users and practitioners. The vulnerability risks prediction model focuses on external characteristics of software.

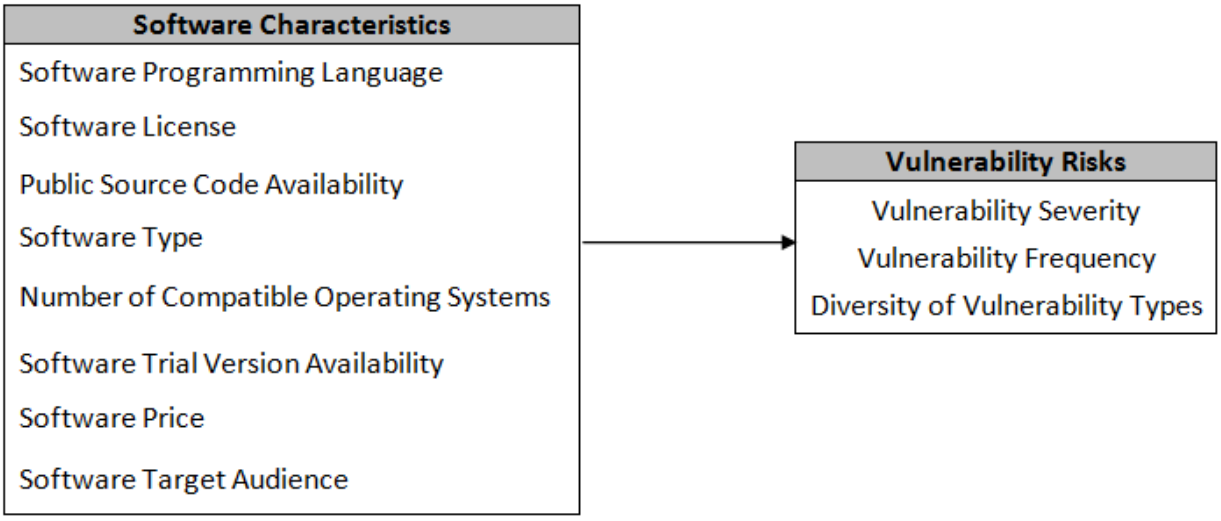


Figure 1: *Vulnerability Risks Prediction Model*

The selection of those characteristics is based on their availability in public databases. For this model (Figure 1), we chose 8 software characteristics namely, software programming language, software license, public source code availability, software type, number of compatible operating systems, software trial version availability, software price and software target audience. Also, since we are predicting vulnerability risks, we chose 3 dependent variables which fall under the umbrella of vulnerability risks. They are vulnerability severity, vulnerability frequency and diversity of vulnerability types. The next section will go over each dependent variable and will be followed by the independent variables and the hypotheses for this model.

2.4 Vulnerability Risks

Vulnerability risk refers to the extent to which vulnerability can have undesirable effects on the security of the software. In this study, we have devised each of vulnerability severity, frequency and type as dependent variables as measures of vulnerability risk. My approach is to assess and predict vulnerability risk of software based on the outcome of each dependent variable. For

instance, software products which have frequent vulnerabilities of different types at high severity levels will be considered to have higher vulnerability risk than software with less frequent, fewer types and low severity levels of vulnerabilities. As discussed earlier, this is an exploratory study and a first look at vulnerability risk from this perspective. The following section discusses each of the dependent variables.

2.4.1 Vulnerability Severity

Vulnerability severity is the extent to which an intruder gains unauthorized privilege on various system resources. Severity is a key attribute since it influences how organizations and businesses react to a specific vulnerability. Oftentimes, vulnerabilities are classified according to their severity levels where each classification is assigned an ordinal or a numerical value. For instance, collecting information about the vulnerable host (i.e. open ports, available services) is considered low-level severity vulnerability. Whereas gaining control of the vulnerable host and compromising the entire system (i.e. remote execution, gaining read and write access to system files) is considered critical-level severity vulnerability.

Throughout literature, various tools and measures have been developed to evaluate and measure severity of software vulnerabilities. The most widely spread and accepted measure is the Common Vulnerability Scoring System (CVSS). According to Schiffman (2007), CVSS is "a universal language to convey vulnerability severity and help determine urgency and priority of response". It's a universal language and a standard for assessing the severity of vulnerabilities in operating systems and software products (Mell et al. 2007). CVSS offers a way to consistently and objectively report on infrastructure vulnerabilities and it solves the issue of multiple,

incompatible scoring systems in use today. Currently CVSS is the most commonly used system by businesses and security firms to assess vulnerabilities and prioritize their tasks, such as creating security patches.

Although there exist numerous other vulnerability severity measures such as SANS Institute, Microsoft and Qualys, it's important to note that CVSS is the only system which provides full details of its equations and how its scores are computed; hence, we will rely on this system as the standard of measure in this study. A more detailed review about other vulnerability severity measures is discussed in this section:

2.4.1.1 SANS Institute

The SysAdmin, Audit, Networking, and Security (SANS) institute has a vulnerability scoring system which is used to assign severity ratings to certain vulnerabilities. The scoring system uses ordinal values (Low, Moderate, and High) rather than numerical scores. Neither the documentation nor the formulas used to generate the scores are available to the public. It's known that SANS generates the ratings by considering several undisclosed factors without weighing them all equally (SANS 2010).

2.4.1.2 Microsoft

Microsoft has a vulnerability rating system known as the Microsoft Security Response Center (MSRC) Security Bulletin Severity Rating System. In this system, each vulnerability is assigned an ordinal value (Low, Moderate, Important or Critical). Similar to the previous rating systems, Microsoft does not reveal the formulas used to determine the ratings (Microsoft 2010).

2.4.1.3 Qualys

Qualys Inc. has a vulnerability rating system which assigns each vulnerability a severity rating between 1 and 5. No additional information is available regarding the rating system (Qualys 2010).

2.4.1.4 CVSS/CVSS 2.0

The first version of CVSS was released in 2005; a more accurate version was later released as CVSS2 in 2007. CVSS2 consists of three sets of measures, namely: base measures, temporal measures and environmental measures. Base measures represent the characteristics of a vulnerability which are constant over time and over user environments. Temporal measures, on the other hand, are the characteristics of a vulnerability which change over time but not among user environments. Environmental measures are the characteristics of a vulnerability which are relevant and unique to a particular user's environment. Each set of measures has a different equation that results in a score which in turn is the essence of CVSS2 metric. The score metric is intended to provide an indication of the relative severity level of the vulnerability. Currently, there are fourteen vulnerability characteristics considered by CVSS2. Such characteristics are split among the three sets of measures mentioned before. For instance, the access vector (AV) metric, part of the base measure, reflects how a vulnerability is exploited (i.e. locally, over an adjacent network, or remotely).

2.4.2 Frequency of Vulnerability

If discovery of vulnerability represents an arrival event, then frequency of vulnerability refers to the rate at which vulnerabilities occurs over a unit of time. According to reports by CERT (2010), there has been an increase in frequency of vulnerabilities coupled with an increase in speed, automation and sophistication of attacks which makes it more difficult for software vendors and system administrators to keep up to date with security patches and they experience more disruptive security attacks. In their work, Arora et al. (2004) studied vulnerabilities and frequency of attacks and looked at vulnerability disclosure policies. While frequency of attacks and vulnerability disclosure have received wide interest among researchers, frequency of vulnerabilities remains a lesser explored topic. The main objective is to measure and analyze vulnerability frequency regardless of vulnerability type. In other words, we are interested in finding the number of vulnerability occurrences within a given time period for different types of software without looking at a specific type or distribution of a vulnerability.

2.4.3 Diversity of Vulnerability Types

Vulnerability type refers to the technical characteristics of a vulnerability. In order to classify vulnerabilities into different types, attack patterns have to be identified in the source code of the vulnerable software. An attack pattern is defined as a "generic representation of a deliberate, malicious attack that commonly occurs in specific contexts" (Moore et al. 2001). Typically, an attack pattern consists of numerous properties such as: the overall goal of the attack, its assumptions, the steps required to carry out the attack and post-conditions of a successful attack (Moore et al. 2001). Once an attack pattern is found, a vulnerability type would be identified and

it would be classified into its proper category. Thereafter vulnerability detection tools can be created to automatically search for and fix the vulnerability (Lowis and Accorsi 2010).

Many software intrusions tools are based on a small number of attack patterns which implies that vulnerabilities can be similar to each other (Kumar and Spafford 1994). For example, there are several known exploits which take advantage of the buffer overflow vulnerability and they all follow the same attack pattern.

A vulnerability type consists of several subtypes which share the same attack pattern but with minor technical differences. Hence, classifying vulnerabilities under the same type (category) ensures that attack patterns are not unnecessarily repeated. For instance, the buffer overflow vulnerability consists of multiple subtypes including stack-based and heap based vulnerabilities. Similarly, Denial of service (DoS) vulnerability has over 10 subtypes including: distributed DoS, reflected DoS, blind DoS, etc. If an attack pattern is unique and different from existing patterns, then chances are it's a new exploit and it can be categorized under a new type of vulnerability. Currently, there is wide range of vulnerability classification frameworks which catalog vulnerabilities according to different attributes. In his study, Meunier (2007) discussed two kinds of vulnerability classifications: popular and scientific. Popular classifications are considered useful and powerful in terms of identifying what went wrong, but they fail to meet scientific criteria. Popular classifications can suffer from the issues of ambiguity and overlapping of vulnerability types. Scientific classifications on the other hand meet rigorous scientific criteria such as reproducibility, objectivity, and lack of ambiguity (Meunier 2007). The next section

reviews some of the existing vulnerability classifications and taxonomies and discusses the limitations of each one.

2.4.3.1 Vulnerability Classification by Software Development Life Cycle (SDLC)

This type of vulnerability classification groups vulnerabilities based on when they were introduced during the software development lifecycle (SDLC). There are numerous types of SDLC models and they differ depending on the software. For instance, the waterfall SDLC model is well known model which consists of seven stages: planning, requirements analysis, design, implementation and development, integration and testing, deployment, and maintenance (Hoffer et al. 2002). The problem with this type of vulnerability classification is that different software employs different SDLC models or other development methodologies. In other words, vulnerability classification depends on the specific SDLC model of the software (Meunier 2007). For instance, one application software development might utilize the waterfall SDLC model while another one might utilize the spiral SDLC model. Therefore, the details of the vulnerability classification are dependent on the SDLC model adopted. This further complicates the process of comparing and classifying vulnerabilities across SDLC models.

2.4.3.2 Vulnerability Classification by Genesis, Time and Location

One of the most recognized works in vulnerability classification is by Avizienis et al. (1994). In their study, the authors classified vulnerabilities based on genesis (How the vulnerability entered the system), by time (When did the vulnerability enter the system) and by location (Where is the vulnerability located inside the system). Each of these dimension are further divided into classes.

Vulnerabilities by genesis are divided into intentional and inadvertent vulnerabilities. Furthermore, vulnerabilities by time are divided into development, maintenance, and operation classes. And finally, vulnerabilities by location are divided into software and hardware. In his book, McGraw (2006) identified that the main advantage of using this type of vulnerability classification is the ability to identify remedy strategies for security problems. For example, if the majority of vulnerabilities are entered inadvertently, then increasing code reviewers and testers can be an effective strategy to resolve the security issue. However, several people including Howard (1997), McGraw (2006) Krsul (1998), and Lindqvist and Jonsson (1997) criticized this framework. Howard (1997) criticized the model for its inability to classify all types of vulnerabilities. For instance, if it's not possible to identify how a vulnerability entered a software, then it's not possible to classify it by genesis. In addition to that, the classes used in this classification are too broad and outdated which means that classifying vulnerabilities is not specific and therefore it's possible to classify a vulnerability into more than one class (McGraw 2006, Krsul 1998). And last, Lindqvist and Jonsson (1997) mentioned that this classification is source code oriented, which means that it's required to have a direct access to the source code in order to classify vulnerabilities by genesis.

2.4.3.3 Aslam's Security Fault Taxonomy

In their work, Aslam (1995) and Aslam et al. (1996) developed a taxonomy which analyzed security faults in UNIX operating system. Faults were classified as into two categories: coding faults and emergent faults. Coding faults were the outcome of design errors and logical programming errors, whereas emergent faults were the result of incorrect installation and

improper administration of the software. One of the problems identified in this taxonomy is the selection criteria of fault classification (Bishop and Bailey 1996). The selection criteria proposed by Aslam (1995) is ambiguous and fault classification depends on the perspective of the classifier which means that it's possible to classify security faults into different categories.

2.4.3.4 Common Weaknesses Enumeration (CWE)

Currently, the most adopted work in vulnerability classification is by Martin et al. (2006). Backed by the Department of Homeland Security as well as the security industry and over 40 organizations from the commercial sector, the authors constructed the CWE taxonomy. This work was built based on publicly available databases and taxonomies such as the Common Vulnerability and Exposures (CVE) database and taxonomies by Aslam (1995) and Tsipenyuk et al. (2006). Martin et al. (2006) created 28 different vulnerability categories containing 290 types of vulnerabilities. The CWE is a currently evolving community standard and it includes enough breadth and detailed granularity which meets requirements of practitioners and security researchers. The CWE exists as a hierarchical tree structure which branches down into the detailed about particular vulnerabilities. Unlike other existing taxonomies, CWE relies on exploring all vulnerability branches and expanding them to find more details about each vulnerability. The CWE taxonomy has been modified and extended by Martin and Barnum (2008) and it's now part of major vulnerability databases such as the National Vulnerability Database (NVD).

Within CWE taxonomy, each classification of vulnerability contains the classification name and an ID. For instance [Cross-Site Scripting 811], cross-site scripting is the classification name and

811 is the ID given to this classification. For the purpose of this study, we will rely on the CWE taxonomy to classify each vulnerability type so that the analysis will have a standard naming scheme.

2.5 Software Characteristics

2.5.1 Software Programming Language

Selecting a suitable programming language is one of the most important decisions which have to be made during software planning and design. A chosen programming language has direct effect on how software ought to be built and what means must be used to guarantee that the software functions properly and securely. Software programs which are written using an insecure language may cause system dependent errors which are known to be difficult to find and fix (Hoare 1973). For example, buffer overflows vulnerabilities and other low-level errors are well known issues which caused major problems in C and C++ languages (Cowan 1999).

As of today, there exist numerous programming languages but the topic of security in programming languages has been widely disregarded as it's believed that programming errors and flaws are caused by numerous factors and cannot be easily eliminated. Current approaches to this issue are essentially ad hoc where best programming practices and secure programming techniques are implemented during or after the design stage. Although this approach helps in preventing coding errors and flaws by relying on programmers' skills and experience, it is hard to say with any certainty what vulnerabilities are prevented and to what extent. More importantly, the ad hoc approach does not protect against new and evolving vulnerabilities as it only handles known vulnerabilities and specific coding flaws.

In his paper, Hoare (1974) stated that a programming language is secure only if the compiler and run time support are capable of detecting flaws and violations of the language rules. The main issue with this statement is that current compilers and debugging tools are not reliable since they parse code differently; therefore, it's impossible to guarantee the same results for the same program. Additionally, such tools do not help programmers in finding vulnerabilities or flaws as they are designed to report syntax errors and limited logical errors. Typically, compilers and debugging tools do not allow for security checks on debugging runs, therefore they are not reliable tools for software security.

An evolving trend in secure programming has been the use of formal language semantics. Formal language semantics try to reason with and prove security properties of the code. For example, in their paper, Leroy and Rouaix (1998) developed a formal technique to validate a typed functional language to ensure that memory locations always contain appropriate values to avoid buffer overflow vulnerabilities. Although the use of formal language semantics has been advocated (Dean et al. 1996, Meseguer and Talcott 1997, Volpano 1997), it has not been adopted as a standard by programmers.

When it comes to software languages, security is dependent on numerous factors such as language developers, programmers and debugging and run-time tools. Without a compelling argument that suggests a likely effect of software language on vulnerability risks, we anticipate a statistically insignificant association.

Hypothesis 1a:

H_0: There is no association between vulnerability severity and software programming language.

H_a: There is an association between vulnerability severity and software programming language.

Hypothesis 1b:

H_0: There is no association between vulnerability frequency and software programming language.

H_a: There is an association between vulnerability frequency and software programming language.

Hypothesis 1c:

H_0: There is no association between diversity of vulnerability types and software programming language.

H_a: There is an association between diversity of vulnerability types and software programming language.

2.5.2 Software License

The diversity of the software business model drives the need for different types of software licenses. A software license is a legal agreement forming a binding contract (relationship) between the vendor and the user of a software product and it's considered an essential part in the evolution of the software to a market product. Software license is regarded as one of the fundamentals of software as there are currently close to 73 different licenses (Perens 2009). Most OSS licenses are classified based on the restrictions they impose on any derivative work (Lerner

and Tirole, 2005). Examples of OSS licenses include GPL, LGPL and BSD. General Public License (GPL) is currently the most popular OSS license which states that any derived work from other GPL software has to be distributed under the same licensing terms. The Lesser GPL (LGPL) and the Berkeley Software Distribution (BSD) are other popular alternatives to GPL with similar characteristics (Comino 2005).

OSS projects rely heavily on code reuse as shown by DrDobbs (2009). In their work, 1311 OSS projects were analyzed and 365,000 instances were found of code reuse among those projects. In principle, most of the OSS licenses allow programmers to modify and reuse existing code. This degree of code inheritance can have positive and negative effects on the security of the software. In their work, Brown and Booch (2002) discussed how reuse of OSS code can inherent insecurities and talked about the concerns which companies have regarding OSS code, how it was developed, and in particular the origins and the reuse of its code. Indeed an analysis study by Pham et al. (2010) suggested that one of the key causes of vulnerabilities is due to software reuse in code, algorithms/standards, or shared libraries/APIs. They proposed the use of new model which uses algorithm to map similar vulnerable code across different systems, and use the model to trace and report vulnerabilities to software vendors. Reuse of OSS software has caused concerns as developers might inherent vulnerabilities from existing code but regardless of the open source community or software vendors' positions on this debate, the possibility of security issues by reusing OSS code has been sufficient to the point where some vendors stopped reusing OSS code in their software. From a security perspective and when it comes to reusing OSS, vendors tend to follow one of the following approaches. Abandon OSS software; only reuse code which has been extensively reviewed; or maintain a relationship with the OSS community and

get involved with the development process (Brown and Booch 2002).

It's my belief that different software licenses will be susceptible to vulnerabilities meaning that there will be a positive association between software licenses and vulnerability risks.

Hypothesis 2a:

H_0: There is no association between vulnerability severity and software license.

H_a: There is a positive association between vulnerability severity and software license.

Hypothesis 2b:

H_0: There is no association between vulnerability frequency and software license.

H_a: There is a positive association between vulnerability frequency and software license.

Hypothesis 2c:

H_0: There is no association between diversity of vulnerability types and software license.

H_a: There is a positive association between diversity of vulnerability types and software license.

2.5.3 Public Source Code Availability

Security of open source software (OSS) and closed source software has been a hot topic with many arguments repeatedly presented. Advocates of OSS argue that more reviewers strengthen the security of the software as it eases the process of finding bugs and speeds it up "given enough eyeballs, all bugs are shallow" (Raymond and Young 2000). Opponents of this idea disagree and claim that not all code reviewers and testers have enough skills and experience compared to code reviewers at companies who are more skilled at finding flaws. The argument is that oftentimes code reviewers and testers need to have further skills other than programming such as

cryptography, stenography and networking. Moreover, proponents of closed source software claim that security by obscurity is the main strength of closed source software since it's harder to find vulnerabilities when the code is not accessible. However, proponents of OSS argue that it's possible to gain access to closed source code through publicly available patches and disassembling software (Tevis 2005).

It's important to note that the impact of source code availability on security depends on the open source development model. For instance, the open source cathedral model allows everyone to view the source code, detect flaws/bugs/vulnerabilities and open reports; but they are not permitted to release patches unless they are approved by project owners. OSS projects are typically regulated by project administrators who require some time to review and approve patches. Attackers can take advantage of the availability of source code and published vulnerability reports to exploit them (Payne 2002). However, proponents of OSS argue that vulnerabilities in OSS projects can be fixed faster than those in closed source software because the OSS community is not dependent on a company's schedule to release a patch. Moreover opponents of this idea argue that the lack of incentives in OSS projects can diminish contributors' motivation to fix vulnerabilities.

Despite the continuous debate on OSS security, advocates from both sides agree that having access to the code makes it easier to locate vulnerabilities but they differ about the impact of vulnerabilities on software security. First of all, keeping the source code open provides attackers with easy access to information which might be helpful to successfully exploit the code. Publically available source code gives attackers the ability to search for vulnerabilities and flaws

and thus increase the exposure of the system. Second, making the source code publicly available does not guarantee that a qualified person will look at the source and evaluate it. In the bazaar style environment, malicious code such as backdoors may be sneaked into the source by attackers posing as trustful contributors. For instance, in 2003 Linux kernel developers discovered an attempt to include a backdoor in the kernel code (Poulsen 2003). Finally, for many OSS projects there is no selection criterion of programmers based on their skills; project owners tend to accept any help without checking for qualifications or coding skills.

Given the issues surrounding source code availability, we hypothesize that making source code publically available will induce attackers and increase vulnerability risks.

Hypothesis 3a:

H_0: There is no association between vulnerability severity and public source code availability.

H_a: There is a positive association between vulnerability severity and public source code availability.

Hypothesis 3b:

H_0: There is no association between vulnerability frequency and public source code availability.

H_a: There is a positive association between vulnerability frequency and public source code availability.

Hypothesis 3c:

H_0: There is no association between diversity of vulnerability types and public source code availability.

H_a: There is a positive association between diversity of vulnerability types and public source code availability.

2.5.4 Software Type

Software can be divided into three major categories: application software, system software, and web applications. Within each category there are dozens, if not hundreds, of specialized software types, but for the purpose of this study, we will concentrate on the most popular software type of each category.

2.5.4.1 Web-Related Software:

According to IEEE Standard (1990), software application refers to software designed to fulfill specific needs of a user. Nowadays, there is a wide range of software applications being developed including word processing programs, database management tools, photo editing software…etc. But during the last decade the web has become the new deployment environment for software applications. Software applications that were previously built for specific operating systems and devices are now being designed specifically for the web (web-enabled). Because of this new movement, and as the web becomes increasingly a universal interface for software development, the software industry is experiencing a major evolution toward web-related software (Festa 2001). For example, the recent release of Google's Chrome web browser which was specifically designed to enable the execution of web applications and services in the web

browser confirms this trend.

As the web evolves, the surrounding technologies are becoming more complex. This is especially relevant in web-enabled applications such as web browsers, email/news clients, VoIP and chat clients which allow the interaction with the web from the client side. Web browsers specifically have become the doorway to the Internet and are currently the most widely used applications and the standard tool for consuming Internet services. This evolution toward web-related applications had a direct impact on the security of such applications. For instance, vulnerabilities and attacks against web browsers became more popular as such attacks compromise the security and privacy and have serious implications for web users. Once a web-related software is infected, the user's web interaction can be fully exposed to the attacker. For instance, an infected web browser can expose the victim's web addresses, forms, user sessions and cookies. Examples of such vulnerability risks against web browsers include key loggers. Key loggers are a form of spyware which can be installed through vulnerability in a web browser and then records all pressed keys when a user visits a certain online website. Moreover, vulnerability risks in a web browser can have a serious implication for intranets (Anupam and Mayer 1998). Since most users use the same browser to access information over the Internet and intranet, a user who has been attacked through a vulnerable web browser would compromise his or her firewall for the duration of the browsing session (Mayer 1998).

The increase in vulnerability risks in web-related software is related to the exponential growth of the Internet. As we enter through the second decade of the 21st century, the rapid adoption of the Internet market along its ubiquitous presence will continue to make Internet technologies such as

web-related applications a prime target for attackers as they constitute the largest mass of victims.

Hypothesis 4a:

H_0: Vulnerability severity of web-related software is less than or equal to system software (operating systems) and web applications.

H_a: Vulnerability severity of web-related software is greater than system software (operating systems) (4a1) and web applications (4a2).

Hypothesis 4b:

H_0: Vulnerability frequency of web-related software is less than or equal to system software (operating systems) and web applications.

H_a: Vulnerability frequency of web-related software is greater than system software (operating systems) (4b1) and web applications (4b2).

Hypothesis 4c:

H_0: Diversity of vulnerability types of web-related software is less than or equal to system software (operating systems) and web applications.

H_a: Diversity of vulnerability types of web-related software is greater than system software (operating systems) (4c1) and web applications (4c2).

2.5.4.2 Web Applications:

The remarkable reach of web applications into all areas of the Internet makes this field among the largest and most important parts of the software industry. As of today, the Internet consists of hundreds of thousands of small and large-scale web applications ranging from e-Commerce

applications to social networking sites to online gaming. This popularity has attracted large user base which made web applications lucrative targets for attackers to exploit vulnerabilities. Web applications are currently subject to plenty of vulnerabilities and attacks, such as cross-site scripting (XSS), session riding (CSRF) and browser hijacking (Mansfield-Devine 2008). Hence, the landscape of vulnerabilities has changed significantly during the first decade of the 21st century. Previously, buffer overflow and format string vulnerabilities accounted for a large fraction of all vulnerabilities during the 1990s, but as web applications became more popular, new vulnerabilities and attacks such as SQL injections and XSS attacks exceeded earlier vulnerabilities. In response, web application vendors dedicated more resources towards securing their products as they tend to receive more attention as possible targets because of their large pool of potential victims (Mercuri 2003). The problem of web application vulnerabilities is becoming more complicated with the recent movement towards Web 2.0 technologies. The landscape of Web 2.0 enables new avenue of vulnerabilities by using sophisticated scripts on the client side. Moreover, Web 2.0 websites are becoming riskier than traditional websites because they use more scripting capabilities to allow users to upload content, share information and gain more control.

Despite the growth of web applications and Web 2.0, these technologies are still limited by the available resources such as network bandwidth, latency, memory and processing power. More specifically, it's argued that web applications are constrained by the capabilities of the web browser they are running in. With this drawback, web application users will ultimately have to rely on their own resources to accomplish magnitudes of tasks. Compared to web-related

software and system software, we hypothesize that web applications will continue to experience vulnerability risks but at a lower rate than other software types.

Hypothesis 4d:

H_0: Vulnerability severity of web applications is greater than or equal to web-related software and system software (operating systems).

H_a: Vulnerability severity of web applications is less than web-related software (4d1) and system software (operating systems) (4d2).

Hypothesis 4e:

H_0: Vulnerability frequency of web applications is greater than or equal to web-related software and system software (operating systems).

H_a: Vulnerability frequency of web applications is less than web-related software (4e1) and system software (operating systems) (4e2).

Hypothesis 4f:

H_0: Diversity of vulnerability types of web applications is greater than or equal to web-related software and system software (operating systems).

H_a: Diversity of vulnerability types of web applications is less than web-related software (4f1) and system software (operating systems) (4f2).

2.5.4.3 System Software:

System software refers to the set of computer programs which are required to support the execution of application programs and maintain system hardware. Operating systems, utilities, drivers and compilers are among the major components of system software. Such components

are the enablers and service providers to software applications. Among these components, the operating system is the most popular and important one. The operating system market has evolved and grown as predicted by theories of network externalities (Shapiro and Varian 1999). But with this growth, there has been a dramatic increase in vulnerabilities. For instance, between 2007 and 2009, the number of operating system vulnerabilities almost doubled from 220 to 420 vulnerabilities (CVE 2010). Such increase in vulnerabilities can be caused by several reasons. First, network externality implies larger user base which makes operating systems an attractive target for hackers. In addition to that, viruses and worms can spread more rapidly because of the large installed user base and network effect. Second, the architecture of some operating systems like Windows allows vulnerabilities to gain a direct access to the operating system files through external scripts; meaning that if malicious scripts are sophisticated enough, they can exploit system files through software applications or through system software directly. And last, the fame factor for discovering vulnerabilities in systems with significant installed user base make them potentially significant target for hackers.

Lately, new technologies such as web-based cloud computing, virtualization and Just enough Operating System (JeOS) have been gradually diminishing the importance of the traditional operating system (Geer 2009). With cloud computing technologies, users can access web applications through their web browser; meaning that an OS like Google Chrome will only be needed to run the web-browser. Moreover, with virtualization technology a personal computer or a server is capable of running multiple OSs at anytime without having the user rely on a single OS. Similarly, Just enough Operating System (JeOS) focuses on running applications which require minimal OS. As these technologies are gaining popularity and becoming more adopted

by users, the role of an OS is starting to decrease so does its network externality. With this is mind, we hypothesize that attackers interests and vulnerability risks will gradually shift to other technologies as they become more popular. Therefore, we believe that vulnerability risks for system software (operating systems) will be less than web-related software and greater than web applications.

2.5.5 Number of Compatible Operating Systems

Software producers often create applications to run on a single or a combination of operating systems (OS). From a software viewpoint, maintaining security is the obligation of both the OS and the software program. But since computer hardware such as the CPU, memory and input/output channels are accessible to a software programs only by making calls to the OS, therefore, the OS bears a tremendous burden in achieving system security by allocating, controlling and supervising all system resources.

For the most part, each of today's streamlines OSs has a main weakness. For instance, earlier OSs such as Windows NT, UNIX and Macintosh had a weakness in their access control policies (Krsul 1998). Such OSs did not specify access control policies very clearly which meant that applications that ran by users inherited all the privileges that the access control mechanisms of the OS provided to those users (Wurster 2010). An access control policy requires an OS to give a program or a user the minimum set of access rights necessary to perform a task. In his work, Denning (1983) illustrated the working of an access control policy which typically consists of three entities namely, subjects, objects and access rights matrix. Subjects refer to users or domains whereas objects are files, services, or other resources and access rights matrix specifies

different kinds of privileges including read, write and execute which are assigned to subjects over objects. A configuration of the access matrix describes what subjects are authorized to do. Vulnerabilities in OSs tend to rely on weaknesses in configuration of access control matrices to gain access to software applications and system software. This creates a serious problem since vulnerabilities can exploit software applications through the OS, gain access and ultimately take over the system.

Moreover, even with an access control policy in place, consideration must be given to system design. The OSs which are in use today have different architectures and are designed with different kernels without considering security and controlled accessibility as significant design criteria. For instance, a large portion of UNIX and Linux vulnerabilities result from boundary condition errors which are commonly known as buffer overflow (Lee and Davis 2003). These boundary conditions result from a failure to check the bound size of arrays, buffers and strings. Attackers tend to exploit this weakness in UNIX and Linux systems to gain access to system software and software applications. Vulnerabilities in Windows OS on the other hand tend to be evenly divided among exceptional conditions, boundary conditions and access control validations (Lee and Davis 2003). With these types of vulnerabilities root break-ins and execution of arbitrary code are common types of attacks.

When it comes to writing software for different platforms, programmers must acknowledge the potential vulnerabilities and threats targeting their software. Since different OSs have different vulnerabilities, the task of designing a secure application tend to become much difficult since programmers have to consider vulnerability risks of each OS. Therefore we hypothesize that:

Hypothesis 5a:

H_0: There is no association between vulnerability severity and the number of compatible operating systems.

H_a: There is a positive association between vulnerability severity and the number of compatible operating systems.

Hypothesis 5b:

H_0: There is no association between vulnerability frequency and the number of compatible operating systems.

H_a: There is a positive association between vulnerability frequency and the number of compatible operating systems.

Hypothesis 5c:

H_0: There is no association between diversity of vulnerability types and the number of compatible operating systems.

H_a: There is a positive association between diversity of vulnerability types and the number of compatible operating systems.

2.5.6 Software Trial Version Availability

Free trial strategy is used by many vendors to promote and sell their goods. This strategy is especially popular and found to be effective to promote and sell digital goods such as software and music. Unlike physical goods, the intangibility of digital products prevents consumers from assessing the products before the consumption and adoption (Heiman and Muller 1996). Such uncertainty of product functionality reduces consumers' motivation to adopt the product and is

considered a source of market failure. Nowadays, offering software free trial at a low marginal production cost has resulted in the prevalence of free trials strategy.

For the software market, there are two strategies of free trial, namely a fully functional free version with limited trial period (time locked version) and a limited functional version (demo version). Each of these strategies has its own advantages and disadvantages. For instance a demo version has an advantage of capturing the network effect from both buyers and trial users. In contrast, some consumers may find it adequate to use only the limited functionalities provided in the demo version rather than purchasing the full version software. Similarly, offering time locked software version can negatively affect the software vendor as consumers with limited usage can utilize this short-term to fully take advantage of the free trial without buying the full software product.

Based on these trial strategies, there have been numerous studies regarding the effect of free trial on software learning curve (Heiman and Muller 1996), software piracy (Chellappa and Shivendu 2005) and software performance (Lee and Tan 2007). For this study, we are interested in measuring the effect of free trial strategies on software vulnerabilities. Although software vendors often release demo or time locked versions, such versions can still contain good source of information for the attackers. Attackers typically misuse the trial versions to look for, find and exploit vulnerabilities. Furthermore, attackers can reverse engineer the limited code and find vulnerabilities (Sutherland et al. 2006). This technique has become particularly important as the attacker can apply vulnerabilities found in free trial versions to exploit full version software. Moreover, there are many hacker groups on the internet who specialize in cracking free trial and

full versions software and releasing them on the internet under what is known as warez. Such groups usually compete with one another to be the first to crack and release the new software. These cracked versions (warez) can also serve as potential targets for attackers looking for vulnerabilities. Hence, while providing free trial versions by software vendors is a marketing strategy, vendors should also expect such free versions to become targets for vulnerabilities and exploits.

Hypothesis 6a:

H_0: There is no association between vulnerability severity and software trial version availability.

H_a: There is a positive association between vulnerability severity and software trial version availability.

Hypothesis 6b:

H_0: There is no association between vulnerability frequency and software trial version availability.

H_a: There is a positive association between vulnerability frequency and software trial version availability.

Hypothesis 6c:

H_0: There is no association between diversity of vulnerability types and software trial version availability.

H_a: There is a positive association between diversity of vulnerability types and software trial version availability.

2.5.7 Software Price

Software price plays an important role in modifying the individual's attitude toward the software in many ways. For example, research studies which looked at software piracy found that software price to be a significant factor (incentive) which influenced the intention to pirate (Gopal and Sanders 2000). In their work, Peace et al. (2003) conducted a survey of 201 respondents and found that software price was among the major reasons for illegally copying software. Following the same analogy, studies have shown that attackers' attitudes and hackers' motivations for finding vulnerabilities are associated with several factors such as: peer approval, self esteem, politics, publicity, financial gains, curiosity and sabotage (Shaw et al. 1999).

Within the hackers' community, hacking achievements typically help individuals gain higher and more respectable status as it refers to the persons' skills and mastery level. Reaching a higher status is oftentimes associated with noteworthy achievements such as hacking popular software. For those hackers who seek publicity or peer approval, they tend to target software with large user base due to their significant reach. So despite software price, hackers look for vulnerabilities in open source and proprietary software as long as there is a significant user base. Similarly, infamous social networking sites such as Facebook and Myspace are constant targets of vulnerabilities regardless of their service cost. Outside the hacker's community, hackers' incentives tend to vary among political reasons (example: Google-China Hacking 2010), financial gains (example: ransom money attacks), self esteem and sabotage. Again, by analyzing each incentive, we find that software price does not likely play any role in vulnerability risks. Hence, there is an expectation of an insignificant relationship. We therefore hypothesize that:

Hypothesis 7a:

H_0: There is no association between vulnerability severity and software price.

H_a: There is an association between vulnerability severity and software price.

Hypothesis 7b:

H_0: There is no association between vulnerability frequency and software price.

H_a: There is an association between vulnerability frequency and software price.

Hypothesis 7c:

H_0: There is no association between diversity of vulnerability types and software price.

H_a: There is an association between diversity of vulnerability types and software price.

2.5.8 Software Target Audience

There are many different types of computer users with a wide range of background, skills, and learning habits. Computer users are typically classified into two distinct groups, namely sophisticated and novice (unsophisticated) users. Sophisticated users have an advanced understanding of computers and Internet technologies; they tend to be more security-aware. Novice users on the other hand refer to non-technical personnel who are not experienced with computers and the Internet; they rely on computers for simple tasks such as word-processing, spreadsheets, and occasional web surfing. Such users are more prone to security issues due to their inexperience. For instance ignoring software updates and security patches, failing to run essential protection utilities such as an anti-virus or firewall applications are typical security issues with novice users. Because of differences in experience level between both groups, some argue that vulnerabilities affect novice users more than sophisticated ones. Although this might be true for viruses and worms and old vulnerabilities, but when it comes to dealing with zero-day

vulnerabilities everyone becomes a victim regardless of their sophistication level. Zero day vulnerabilities refer to unreported exploitable vulnerabilities for which a patch is not available from software vendors. Moreover, when it comes to attackers and potential targets, eventually everyone is a target. Despite the type of computer users, the objective of vulnerability attacks is to hack as many computers with the least possible effort (Spitzer 2002). Attackers tend to focus on a single vulnerability and use automated scanning tools to search for as many systems as possible for that vulnerability. Such automated tools are often called autorooters and can be designed to scan a specific network for vulnerable machines or scan a range of IP addresses until a victim is found. It's important to note that these tools do not distinguish between software users as they look for any vulnerable target in sight. Given the nature of vulnerability attacks, we therefore hypothesize the following with an expectation of an insignificant relationship:

Hypothesis 8a:

H_0: There is no association between vulnerability severity and software target audience.

H_a: There is an association between vulnerability severity and software target audience.

Hypothesis 8b:

H_0: There is no association between vulnerability frequency and software target audience.

H_a: There is an association between vulnerability frequency and software target audience.

Hypothesis 8c:

H_0: There is no association between diversity of vulnerability types and software target audience.

H_a: There is an association between diversity of vulnerability types and software target audience.

CHAPTER 3 Research Method

3.1 Research Design and Methodology

The research design of this study consists of two parts: primary and secondary analysis of vulnerability risks. This approach allows us to study vulnerability risks from objective and subjective sources. It also extends the knowledge on how vendors and practitioners predict vulnerability risks, meanwhile, gain better understanding of the relationship between software characteristics and vulnerability risks. Furthermore, for practitioners, gaining a better understanding of this relationship will be important as it can be used as predictors for better software security.

3.2 Survey Design

Through the review of existing vulnerability literature we were unable to find a suitable survey instrument that had been validated and fits the objectives of this study. Therefore, in order to address these research questions, a cross-sectional survey instrument was developed and used to collect information from IT practitioners. Because of their role in the software business, IT practitioners are in a good position to answer questions dealing with vulnerability risks and thus they are ideal subjects for this study. There are several reasons why I believe this study did not suffer from flaws related to instrument validity. First, the survey was written in simple language based on software terminology which is widely used and understood among IT practitioners. Second, members of the research committee reviewed the questionnaire in order to determine whether survey questions and instructions were clear, understandable and accurately assess the

main objective. This helped to eliminate potential problems and to promote a higher response rate. Committee members were able to report unclear questions, the length of time needed to finish the survey and any problems encountered. Based on the results, the appropriate modifications were made to the survey instruments to clarify areas of concern. Once survey instruments were finalized, I followed the validated strategies to maximize response rates (Dillman 2007). A two contact email strategy was used. First, an email was sent to invite respondents to participate in the survey. A second email was sent after 10 days as a reminder to non-respondents with a link to the survey.

3.3 Survey Analysis

In order to measure the effects of software characteristics on vulnerability risks, I used descriptive statistics for several reasons. First, the design of the survey questionnaires did not match the conditions necessary to employ sophisticated statistics. Second, the survey questions were best answered with descriptive statistics since these questions dealt with the degree to which respondents agreed/disagreed with the statements. Therefore, these questions were best analyzed with counts, percentages and confidence intervals.

3.4 Vulnerability Data Collection

Data gathering for the independent variables was based on a random sample of software products drawn from a list of three software types, namely web-based software applications, operating systems and web applications. For each software product, we referred to software vendor's website and publically available information on the web to obtain data regarding the following software characteristics:

- Software programming language
- Software license
- Public source code availability
- Software type
- Number of compatible operating systems
- Software trial version availability
- Software price
- Software target audience

In order to obtain data for the dependent variables, we used vulnerability data available in the National Vulnerability Database (NVD). NVD is created by the National Institute of Standards and Technology (NIST) and sponsored by the US government. NVD contains publicly known security vulnerabilities of software and hardware products aggregated from security firms, organizations, forums and advisory groups. It's considered one of the most reliable and comprehensive vulnerability databases on the web. As of November 2010, NVD contained approximately 44,305 vulnerabilities collected from 1999 to 2010. NVD works in compatibility with numerous sources including CVE, CPE, CWE and CVSS to enhance and standardize vulnerability classifications. The database provides vulnerability information searches, RSS feeds and vulnerability database downloads in CSV, text, and XML formats. Despite the large amount of information in NVD, for the purpose of this study, we were interested only in the following

- The unique name and ID for the software

- The diversity of vulnerability types
- Publication date of vulnerabilities
- The vulnerability severity scores (CVSS)

3.5 Vulnerability Data Analysis

In order to examine the relationship between vulnerability risks and software characteristics, three regression models were built: vulnerability severity model, vulnerability frequency model and diversity of vulnerability types model. Each model included eight independent variables namely, software programming language, software license, public source code availability, software type, number of compatible operating systems, software trial version availability, software price, and software target audience. In order to build the models, the research used the proper coding for each independent variable as stated below. Also, prior to running the analysis, all the required assumptions were tested for each model.

Table 1 and Table 2 below contain information about the independent and dependent variables respectively along with information regarding variable types, coding and descriptions.

Table 1:
Dependent variables of vulnerability risks

Variable Name	*Variable Type*	*Variable Descriptions & Coding*
Vulnerability Severity	Ratio	The average severity level of all vulnerabilities relating to a software product.
Vulnerability Frequency	Ratio	The rate at which vulnerabilities occurred over a month starting from the date of first published vulnerability until the date of the last published one.
Diversity of Vulnerability Types	Ratio	A count of the number of unique vulnerability types of a software.

Table 2:

Independent variables of vulnerability risks model

Variable	*Variable Type*	*Variable Description and Coding*
Software Programming Language	Nominal	Indicates the type of programming language used to develop the software. Software programming languages was divided into two groups (Static Languages=0, Dynamic Languages=1) according to the programming paradigm used.
Software License	Ordinal	Refers to the type of software license. Software license was divided into 4 categories ranging from completely unrestricted license to completely restricted license. Dummy variables were used in the model.
Public Source Code Availability	Nominal	This variable indicates whether software source code is open, closed or mixed. Dummy variables were used in the model.
Software Type	Nominal	Refers to the type of the software product. (Web-based software, Operating Systems, Web applications). Dummy variables were used in the model.
Number of Compatible Operating Systems	Ratio	Indicates the total number of operating systems which are compatible with the software product.
Software Trial Version Availability	Nominal	0=A trial version is available 1=A trial version is not available
Software Price	Interval	This variable considers the price of a single software license
Software Target Audience	Nominal	Software target audience is characterized by the level of sophistication needed to operate/use a software product. (0= Sophisticated users, 1= Sophisticated and Unsophisticated users)

Chapter 4 Results and Conclusion

4.1 Survey

4.1.1 Survey Questionnaire Development

All questions in the survey, except the demographic questions, related directly to the stated research objectives. A panel of experts which included members of the dissertation committee assessed the face and content validity of the questionnaire. The panel commented on the design, format and wording of the survey. Such comments and suggestions were noted and addressed accordingly. After the questionnaires were corrected, the final approved survey was used to gather the actual data essential for this study.

Survey questionnaires were grouped into 5 sections (see Appendix A). Section 1 addressed demographic questions, including participants' job title, their organization/sector and their security knowledge. Section 2 consisted of 8 questions which assessed the effect of software characteristics on vulnerability risks using a 7-point likert scale. The points on the scale ranged from 7 "strongly agree" to 1 "strongly disagree" and they were consistent with the original form of the scale (Westaway and Maluka 2005). Section 3 asked participants to rank the top three software characteristics that have the most effect on vulnerability risks. Finally, sections 4 and 5 were reserved for any additional comments respondents might have regarding software characteristics or the overall study.

4.1.2 Data Collection of Survey

In order to collect data reflecting the opinion of IT practitioners, an online survey was created. The online survey was used to reach a large number of professionals in a timely and efficient manner. A list of email addresses was obtained from online business directories; the list contained 2,581 entries of IT practitioners. Information was then entered into an online mailing application called "Simple Mailing List", and an invitation to participate was sent (see **Error! Reference source not found.** for the email). The invitation email was sent containing information and a hyperlink. Of these emails, 202 email addresses were invalid leaving a total of 2,379 emails sent. A reminder email was sent approximately three weeks later. The survey collected data for 6 weeks with a total of 218 respondents. When participation ceased after 6 weeks, the survey was closed in order to prepare for data analysis.

4.1.3 Preparation and Analysis of Survey Data

According to Cooper and Schindler (2003), data analysis has to do with the reduction of accumulated data into manageable size, development of summaries, and statistical analysis. This section described how the data was prepared and the method of data analysis used in this study.

4.1.3.1 Data Preparation

Survey responses were captured into a SPSS spreadsheet. Questionnaires were checked for completeness. Preliminary data analysis included addressing missing data and ensuring assumptions were met for appropriate data analysis. Missing data were addressed with the appropriate technique. Data were further examined to ensure assumptions were met for the selected statistical procedure.

4.1.3.2 Missing Data

The responses to the study were examined for missing data. Missing data were found in the demographic questionnaires. Of a possible 218 responses, job title had 5.3% missing responses (N=10), work history at current position had 9.6% missing responses (N = 18), security knowledge had 5.9% missing responses (N = 11), industry of organization had 8.6% missing responses (N = 16), and sector of organization had 10.2% missing responses (N = 19). Because such a low number of values were missing for these demographic variables, records with missing data were kept in the sample. The remaining question did not have any missing values since the question did not allow empty responses. Additionally, 14% of the respondents (N=31) started but did not complete the survey; these responses were removed. The final sample size was 187.

4.1.3.3 Summary of Survey Responses

This section provides an overview of the survey results. Section I asked respondents to provide demographic information. The results of section I are show in Table 3, Table 4, Table 5, Table 6 and

Table 7.

Table 3:
Frequency Table of Job Titles

	Frequency	Percent
No response	10	5.3

Administrative / Office Manager	13	7.0
Data Analyst	10	5.3
Database Administrator (DBA)	11	5.9
Network Administrator	10	5.3
Network Engineer	9	4.8
Office Administrator	6	3.2
Programmer Analyst	10	5.3
Project Engineer	10	5.3
Project Manager	11	5.9
Software Developer	9	4.8
Software Engineer	10	5.3
Support Technician	13	7.0
Systems Administrator	15	8.0
Test / Quality Assurance	10	5.3
Web Designer	11	5.9
Web Developer	8	4.3
Other	11	5.9
Total	187	100.0

Table 4:
Frequency Table of Work History at Current Position

	Frequency	Percent
No response	18	9.6
Less than 1 year	24	12.8
1 - 2 years	18	9.6
2 - 5 years	19	10.2
5 - 10 years	31	16.6
10 - 15 years	29	15.5
15 - 20 years	24	12.8
Over 20 years	24	12.8
Total	187	100.0

Table 5:
Frequency Table of IT Security Knowledge

	Frequency	Percent
No response	11	5.9
Beginner	52	27.8
Intermediate	68	36.4
Expert	56	29.9
Total	187	100.0

Table 6:
Frequency Table of Industry of Organization

	Frequency	Percent
No response	16	8.6
Accommodation and Food Services	5	2.7
Administrative and Support and Waste Management	8	4.3
Agriculture, Forestry, Fishing and Hunting	3	1.6
Arts, Entertainment, and Recreation	12	6.4
Construction	10	5.3
Educational Services	8	4.3
Finance and Insurance	4	2.1
Health Care and Social Assistance	10	5.3
Information	16	8.6
Management of Companies and Enterprises	12	6.4
Manufacturing	13	7.0
Mining, Quarrying, and Oil and Gas Extraction	7	3.7
Professional, Scientific, and Technical Services	8	4.3
Public Administration	9	4.8
Real Estate and Rental and Leasing	6	3.2
Retail Trade	12	6.4
Transportation and Warehousing	6	3.2
Utilities	5	2.7
Wholesale Trade	7	3.7

Other	10	5.3
Total	187	100.0

Table 7:
Frequency Table of Sector of Organization

	Frequency	Percent
No response	19	10.2
Public sector	41	21.9
Private sector	94	50.3
Not-for-profit	20	10.7
Other	13	7.0
Total	187	100.0

Figure 2 indicates the respondents' opinions in reference to the statement "Vulnerability risk is affected by software type". One hundred and forty five respondents (77.54%) agreed with the statement, 28 respondents (14.97%) disagreed, and 28 respondents (14.97%) were neutral.

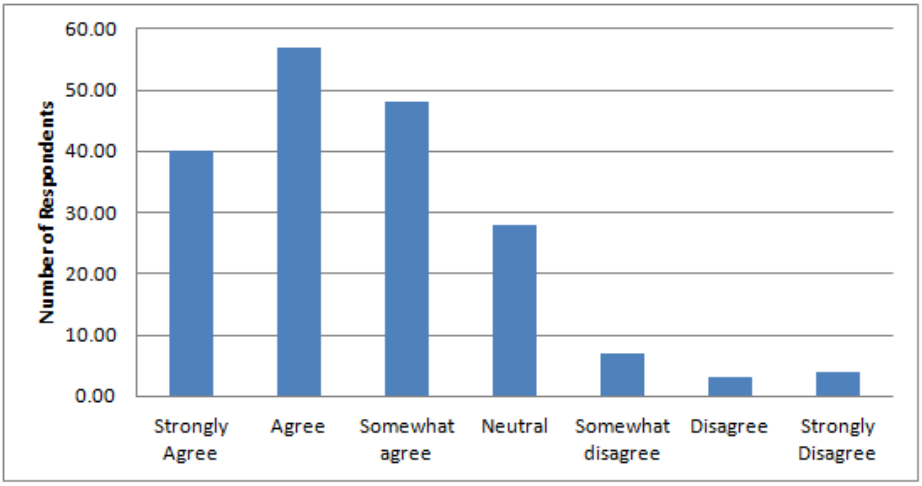


Figure 2: Responses to the Statement "Vulnerability risk is affected by Software Type"

Figure 3 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by software programming language". Forty two respondents (22.46%) agreed with the statement, 117 respondents (62.57%) disagreed, and 28 respondents (14.97%) were neutral.

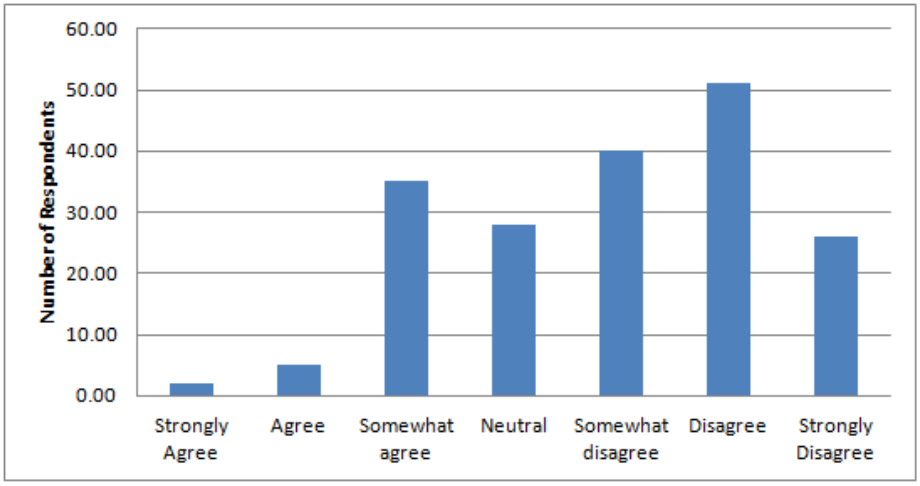


Figure 3: Responses to the Statement "Vulnerability risk is affected by Software Programming Language"

Figure 4 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by public source code availability". Eighty five respondents (45.54%) agreed with the statement, 76 respondents (40.64%) disagreed, and 26 respondents (13.90%) were neutral.

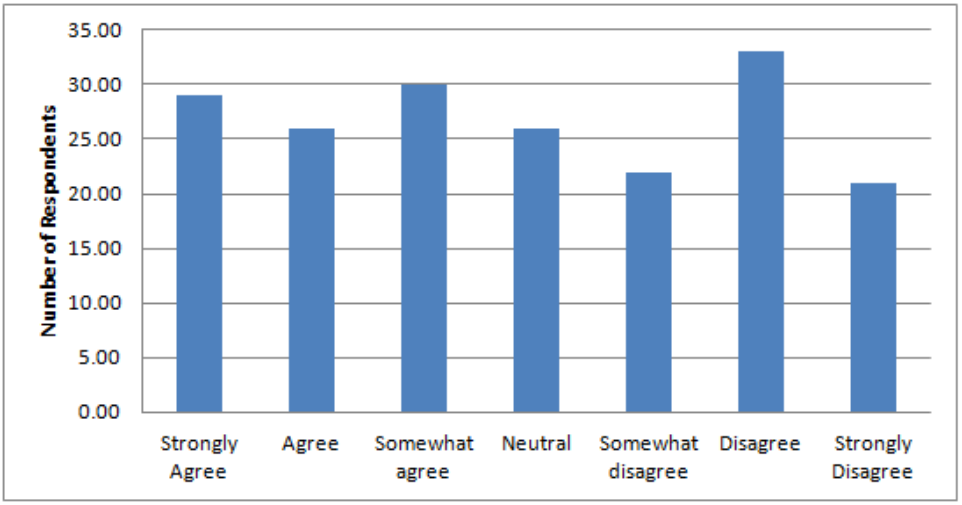


Figure 4: Responses to the Statement "Vulnerability risk is affected by Public Source Code Availability"

Figure 5 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by software license". Twenty five respondents (13.37%) agreed with the statement, 148 respondents (79.14%) disagreed, and 14 respondents (7.49%) were neutral.

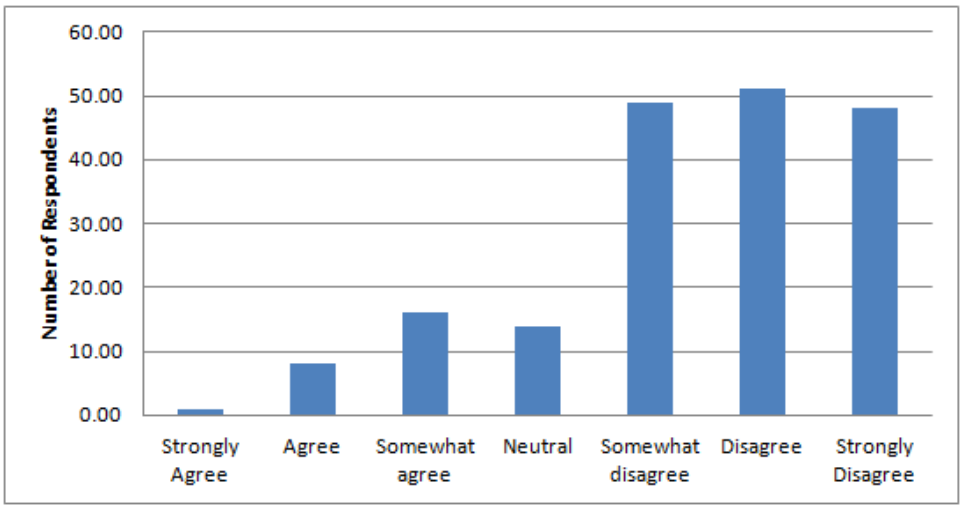


Figure 5: Responses to the Statement "Vulnerability risk is affected by Software License"

Figure 6 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by the number of compatible operating systems". One hundred and thirty eight respondents (73.80%) agreed with the statement, 26 respondents (13.90%) disagreed, and 23 respondents (12.30%) were neutral.

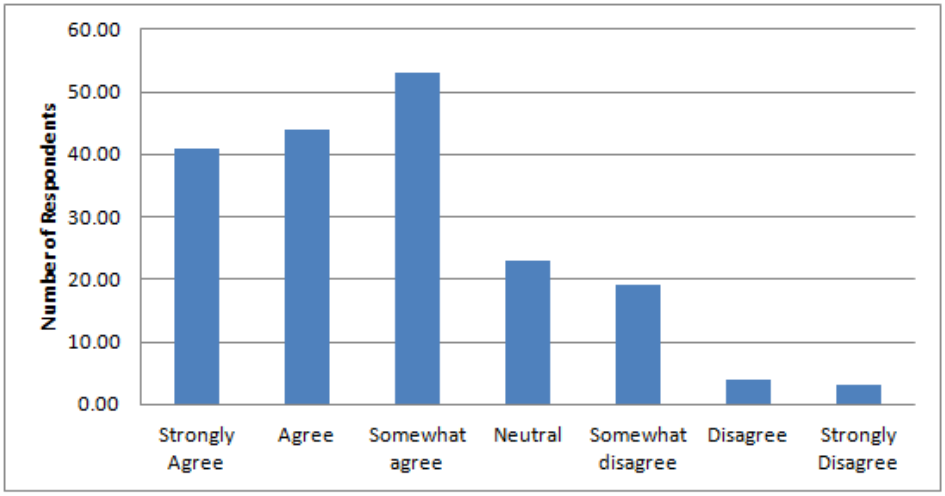


Figure 6: Responses to the Statement "Vulnerability risk is affected by Number of Compatible Operating Systems"

Figure 7 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by software trial version availability ". Fifty two respondents (27.81%) agreed with the statement, 103 respondents (55.08%) disagreed, and 32 respondents (17.11%) were neutral.

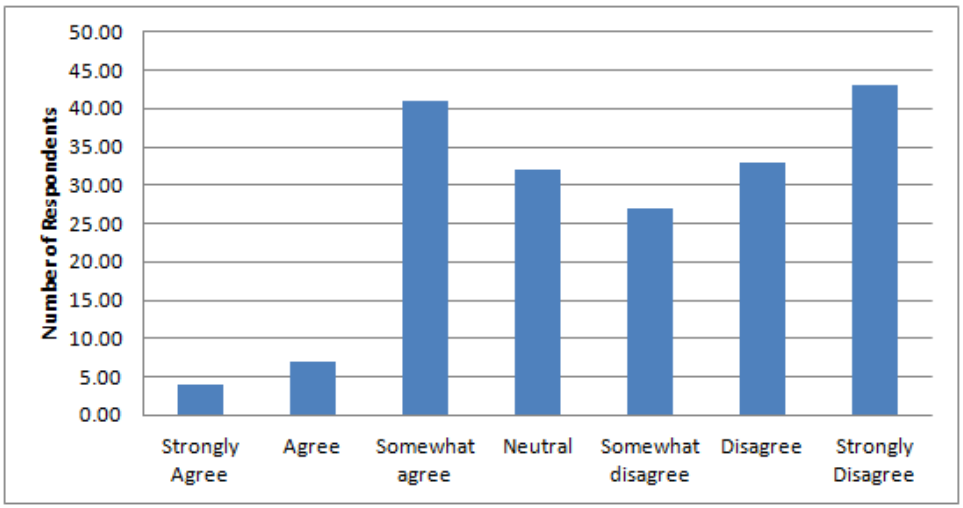


Figure 7: Responses to the Statement "Vulnerability risk is affected by Software Trial Version Availability"

Figure 8 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by software price". Two respondents (1.07%) agreed with the statement, 181 respondents (96.79%) disagreed, and 4 respondents (2.14%) were neutral.

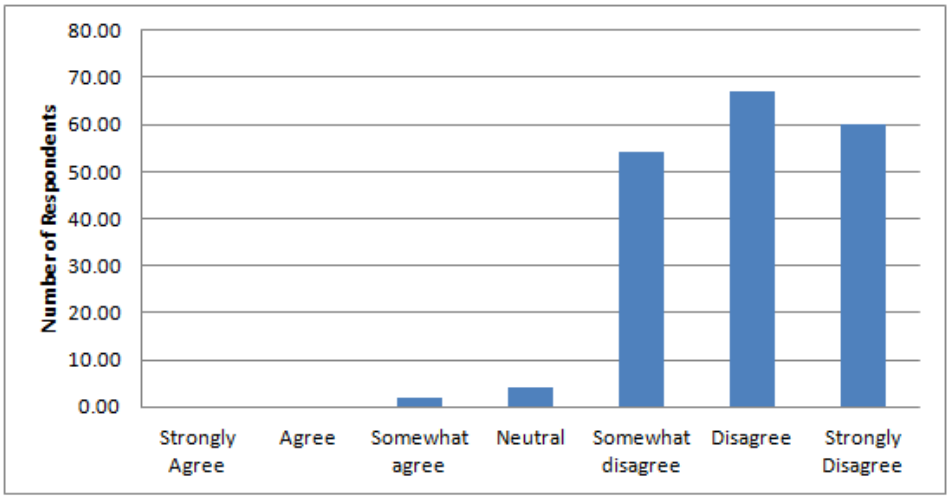


Figure 8: Responses to the Statement "Vulnerability risk is affected by Software Price"

Figure 9 indicates the respondents' opinions when it comes to the following statement: "Vulnerability risk is affected by software target audience ". Fifteen respondents (8.02%) agreed with the statement, 142 respondents (75.94%) disagreed, and 30 respondents (16.04%) were neutral.

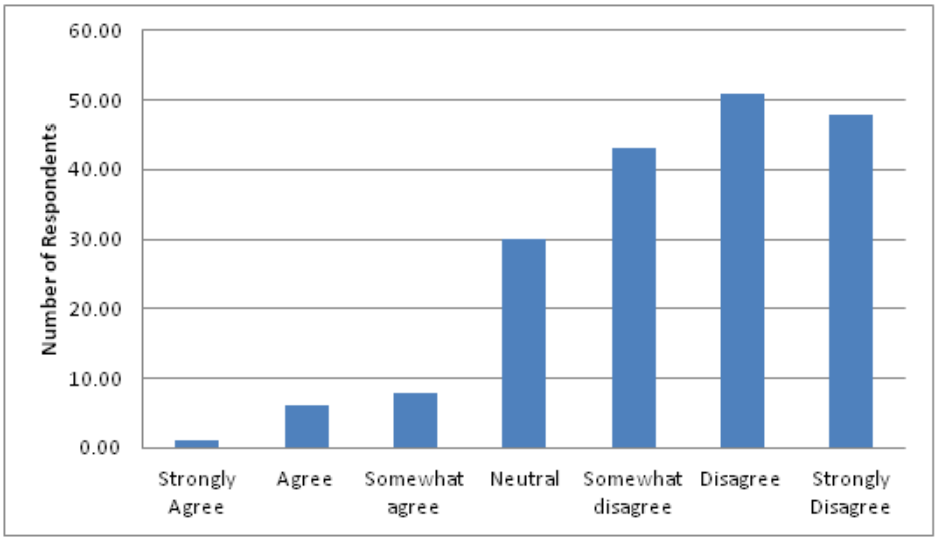


Figure 9: Responses to the Statement "Vulnerability risk is affected by Software Target Audience"

A one-sample t test was conducted to evaluate whether the means of survey responses were significantly different from "4" (neutral response). The significance levels derived from the one sample t-test are included in Table . The results confirm that the significance levels of all survey responses except for one are less than 0.05 indicating that they are statistically different from "4" (neutral response). Survey question concerning Public Source Code Availability had a p-value of .509 which indicates that the response is not statistically different from "4" (neutral response).

Table 8:
One sample t-test results at α = 0.05

	Mean	Std. Deviation	95% Confidence Interval of the Difference		Sig. (2-tailed)
			Lower	Higher	
Vulnerability risk is affected by Software Type	2.63	1.348	-1.57	-1.18	.000*
Vulnerability risk is affected by Software Programming Language	4.90	1.463	0.69	1.11	.000*
Vulnerability risk is affected by Public Source Code Availability	3.90	1.990	-0.38	0.19	.509
Vulnerability risk is affected by Software License	5.39	1.430	1.18	1.60	.000*
Vulnerability risk is affected by Number of Compatible Operating Systems	2.70	1.330	-1.49	-1.11	.000*
Vulnerability risk is affected by Software Trial Version Availability	4.98	1.555	0.75	1.20	.000*
Vulnerability risk is affected by Software Price	5.96	0.891	1.83	2.09	.000*
Vulnerability risk is affected by Software Target Audience	5.44	1.316	1.25	1.63	.000*

* $p < 0.05$

Table summarizes the responses of the survey questionnaires. In this table, responses were grouped into three categories namely, Agree, Disagree and Neutral. Respondents who answered strongly agree, agree or somewhat agree were grouped together and labeled as “Agree”. Also, respondents who answered strongly disagree, disagree or somewhat disagree were grouped together and labeled as “Disagree”. Based on the results, the majority of respondents 145 (77.54%) believed that software type had the most effect on vulnerability risks.

Table 9:
Summary of Survey Responses

	Agree	Disagree	Neutral

Vulnerability risk is affected by Software Type	145 (77.54%)	14 (7.49%)	28 (14.97%)
Vulnerability risk is affected by Number of Compatible Operating Systems	138 (73.80%)	26 (13.90%)	23 (12.30%)
Vulnerability risk is affected by Public Source Code Availability	85 (45.54%)	76 (40.64%)	26 (13.90%)
Vulnerability risk is affected by Software Trial Version Availability	52 (27.81%)	103 (55.08%)	32 (17.11%)
Vulnerability risk is affected by Software Programming Language	42 (22.46%)	117 (62.57%)	28 (14.97%)
Vulnerability risk is affected by Software License	25 (13.37%)	148 (79.14%)	14 (7.49%)
Vulnerability risk is affected by Software Target Audience	15 (8.02%)	142 (75.94%)	30 (16.04%)
Vulnerability risk is affected by Software Price	2 (1.07%)	181 (96.79%)	4 (2.14%)

Respondents were asked to rank the top 3 software characteristics which had the most effect on vulnerability risks. According survey results, 76 (40.64%) of the respondents ranked software type to have the most effect on vulnerability risks. The number of compatible operating systems was ranked in the second position with 59 (31.55%) respondents. Public open source availability was ranked in the third position with 43 (23%) respondents.

4.2 Vulnerability Data

4.2.1 Vulnerability Data Collection

The research population in this study consisted of software products available at the National Vulnerability Database (NVD). According to the NVD, all software products are listed based on the Common Platform Enumeration (CPE) standard. CPE is a naming scheme for software products which includes a formal name format, type of software, and description of supported platforms. CPE is used to ensure that all software products are identified with a unique ID.

In order to obtain the research population, I obtained a copy of the NVD database as an XML file and extracted the names of software products. I used CPE naming scheme in order to exclude any software products which are not part of the application/operating systems categories. Next, I cleaned up the dataset. I then categorized software into three types namely, web-related software, operating systems and web applications. Web-related software included any product which deploys on clients' machine and depends on the web to fulfill specific needs of the software user. For example web browsers, chat clients, VoIP applications, firewalls, RSS readers. Operating systems on the other hand included systems which deploy on personal computers thus excluding systems designed for mobile phones, routers, hubs and other hardware devices. And last, web applications included any application which depends on the web for deployment and correct execution. For instance, forum applications, content management systems (CMS), wikis, web carts.

The next step involved collecting software characteristics for each software product. I used software vendors' website as the main source of information in addition to vendors' support forums and repositories. The obtained information included seven software characteristics: software programming language, software license, public source code availability, the number of compatible operating systems, software trial availability and software price. In order to determine software target audience, I relied on end user knowledge as a determiner of software target audience. Basically, software products which require end users to have extensive knowledge to operate the software are labeled as products targeting sophisticated users. On the other hand, software products which do not demand extensive knowledge from end users are

labeled as software targeting both sophisticated and unsophisticated users since both groups can operate the software.

Four graduate students were asked to independently assign values to each software product. Once values were assigned, I checked for inter-rater reliability. In order to ensure high inter-rater reliability and to make sure that assignments and are not dependent on a single rater, I measured inter-rater reliability using Krippendorff's alpha (α). Krippendorff (2004) suggests researchers can generally rely on variables with α of at least 0.8 and treat with caution variables with α less than 0.8 but at least 0.667. The average inter-rater reliability was acceptable with a Krippendorff α of 0.82.

As for the dependent variables, the obtained NVD dataset contained vulnerability information for each software product. Vulnerability severity was determined by calculating the average severity score for each software product. Vulnerability severity scores ranged from 0 to 10, where 0 refers to least severe vulnerability and 10 is most severe vulnerability.

Vulnerability frequency relied on the published date of the vulnerability. In order to calculate vulnerability frequency, I computed the rate at which vulnerabilities occurred starting from the date of first published vulnerability until the date of the last published one. Vulnerability frequency was determined based on a monthly basis. For those products with only one vulnerability, I used the actual month of data collection to mark the end period.
Diversity of vulnerability types was also acquired through the dataset. Diversity of vulnerability types was calculated by counting the number of unique vulnerability types of a software product. Vulnerability types were coded according to Common Weakness Enumeration (CWE) standard.

4.2.2 Vulnerability Data Preparation

Preliminary data analysis included addressing missing data, detecting outliers, and ensuring assumptions were met for appropriate data analysis. Missing data were addressed with the appropriate technique. Data were further examined to ensure assumptions were met for the selected statistical procedure.

4.2.2.1 Missing Data

Missing data was present for dependent variables. Of a possible 257 software products, vulnerability severity had 4.4% missing data points (N = 13), vulnerability frequency had 5.8% missing data points (N=17) and diversity of vulnerability types had 4.1% missing data points (N = 12). Those software products with missing information were removed from the sample.

4.2.2.2 Sampling

Using SPSS, a random sample of software products was drawn for each of the three software categories. For a reliable regression equation, a sample size with a ratio of 15 cases per variable is recommended (Stevens 1996). Among the sample data, software that was categorized in the operating system category was deemed to be inappropriate in this study for the following reasons. First, independence of sample data was an issue since most operating systems were either part of the Windows, Linux or Mac families. This implied that there is a dependency issue among Windows products, Linux distributions and Mac products. For instance, since all Windows products had the same kernel, this meant that Windows vulnerabilities are shared among different versions of its operating systems. Second, with limited number of operating

systems, making comparison with other categories, namely related software and web applications, will be biased (Ryan 2009). The removal of operating systems also implied retracting hypotheses concerning operating systems. After excluding operating systems, the final sample size was 207 software products. Hair et al. (1998) provided that the ratio of observations to the regressors should never be under ratio of 5:1. Therefore with a sample size of 207, this condition was met.

4.2.2.3 Outliers

Outliers are defined as values that are within 3 standard deviations of the mean (Mertler and Vannatta 2005). Outliers can be dealt with in four ways: 1) delete the data points 2) count extreme values as missing but retain data for other variables 3) transform the value 4) reduce the value to be the highest value possible while staying within the three standard deviations (Tabachnick and Fidell, 1989). According to Hair et al. (2006), removal of outliers reduces the probability of Type I and Type II errors and it increases the accuracy of estimates (Osborne and Overbay, 2004). For the study, seven values were above and/or below three standard deviations and were removed from the sample.

4.2.3 Multiple Linear Regression

Multiple linear regression analysis is a mathematical maximization method (Brace et al. 2002). It's the process of predicting the dependent variable based on the association between a dependent variable and two or more independent variables (Ryan 2009). In multiple linear regression, each variable has its own regression coefficient that gives its relative importance in the relationship outcome. According to Brace et al. (2002), multiple linear regression is

appropriate for exploring linear relationships between dependent and independent variables. Also, the dependent variable being predicted should be measured on a continuous scale and the independent variables should be measured on a ratio, interval, ordinal, or nominal scale (expressed using dummy variables). In addition to that, multiple linear regression requires a large number of observations. The number of cases must exceed the number of independent variables that are being used in the regression.

Multiple linear regression analysis involves the creation of a mathematical equation which describes the relationship between the dependent and independent variables (Keller 1997). The prediction of dependent variable (Y) is accomplished by the following equation (Levine et al. 1995):

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_i X_i + \varepsilon$$

Where:

Y = the predicted value of Y for observation

X_i = the matrix of covariates

β_0 = Y intercept when all X_i are set to be zero

β_i = the regression coefficient for variable X_i when all the remaining independent variables X_i ($i \neq j$) are held constant

ε = error

The use of multiple linear regression is appropriate in the study because the dependent variables are continuous. Also, with eight independent variables and a sample size of 207, the ratio of cases per variable was large enough (Hair et al. 1998). Prior to running the analysis,

multicollinearity, normality, linearity, homoscedasticity and independence assumptions were checked using various methods. Tests for these assumptions are discussed later in section 4.2.4.1.

4.2.4 Poisson Regression

Poisson regression is commonly used to model events that occur infrequently in time or space and it's also used for count variables (Cameron and Trivedi 1998). Poisson regression assumes the dependent variable, which follows a Poisson distribution is controlled by independent variables. The regression equation is expressed into a logarithm of the covariates (DeMaris 2004):

$$\log(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_i X_i + \varepsilon$$

or

$$Y = \exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_i X_i + \varepsilon)$$

Where:

Y = the predicted value

Xi = the matrix of covariates

β_0 = Y intercept when all X_i are set to be zero

β_i = the Poisson regression coefficient for variable X_i

ε = random error term

For this study, the choice of using Poisson regression was appropriate for Diversity of Vulnerability Types variable due to its distribution. Diversity of vulnerability types is a count variable which has a Poisson distribution. Before choosing Poisson regression, several

transformation methods were applied to the dependent variable but normality wasn't obtained. Therefore and since Poisson regression does not assume normal distribution of the dependent variable, it was deemed to be appropriate.

Prior to running the analysis, multicollinearity, independence and equality of variance and mean were checked using the proper methods. Tests for these assumptions are discussed in section 4.2.4.2.

4.2.4.1 Assumptions of Multiple Linear Regression

Assumptions of multiple linear regression are important for the representativeness of the results of the sample. The assumptions of the multiple linear regression analysis are: absence of multicollinearity, normality, linearity homoscedasticity and independence.

Multicollinearity occurs when there is close to a near perfect linear relationship among the independent variables in a regression model which means that there is some degree of redundancy or overlap among variables. Multicollinearity causes a loss in power and it makes interpretation more difficult. Since multiple regression is sensitive to multicollinearity, I used two tests for determining multicollinearity namely, tolerance and variance inflation factor (VIF) (Mansfield and Helms 1982). The tolerance of value zero implies that the variable being assessed is completely predictable from the other independent variables; in other words, it is a perfect multicollinearity. A tolerance value of one implies that the variable being assessed is completely not predictable from the other independent variables; in other words, it is a perfect non-multicollinearity. A general rule of thumb is that if tolerance value is lower than 0.2, a multicollinearity problem could occur (Menard 2002). Second, variance inflation factor (VIF)

was used. According to Lomax (2001), VIF refers to the inflation that occurs for each regression coefficient above the ideal situation of uncorrelated predictors. Wetherill (1986) and Belsley et al. (1980) suggest that VIF should be less than 10 in order to satisfy this assumption. After running collinearity diagnostics, multicollinearity was found between Public Source Code Availability and Software License variables. Both variables had a tolerance value < 0.2 and a VIF value greater than 10. Public source code availability was chosen since it had a stronger positive association with the dependent variables than software license. Table 10 illustrates collinearity diagnostics after removal of software license.

Table 10:
Collinearity diagnostics: Tolerance and Variance Inflation Factor (VIF) values

	Collinearity Statistics	
	Tolerance	VIF
Software Programming Language	.703	1.422
Public Source Code Availability Dummy1	.749	1.335
Public Source Code Availability Dummy2	.842	1.187
Software Type	.723	1.383
Number of Compatible Operating Systems	.842	1.188
Software Trial Version Availability	.898	1.114
Software Price	.935	1.069
Software Target Audience	.971	1.030

Normality was checked by the analysis of the normal probability plots of the residuals where the standardized residuals – observed vs. expected – need to follow the normal line to be considered normally distributed. The visual inspection of the P-P plots and the non-significant Shapiro-Wilk

test proved that the regression was normally distributed for Vulnerability Severity ($p > 0.295$) and Vulnerability Frequency ($p > 0.626$) as illustrated in Figure 10 and Figure 11 respectively.

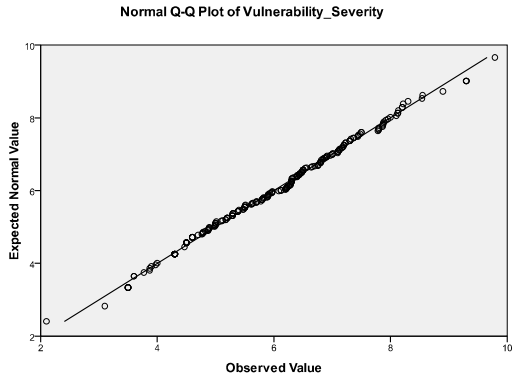


Figure 10: Normal Probability Plot of the standardized residuals for Vulnerability Severity

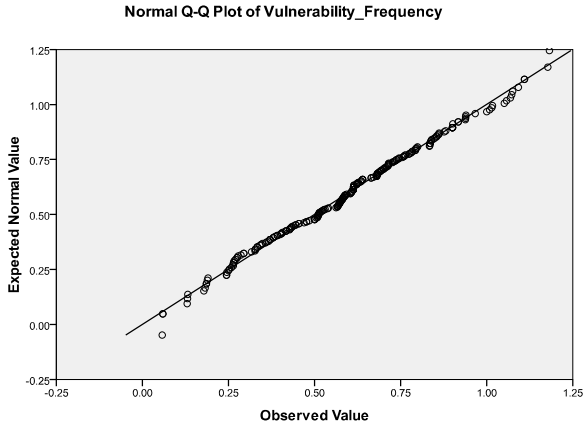


Figure 11: Normal Probability Plot of the standardized residuals for Vulnerability Frequency

Linearity was assessed through the residuals analysis and partial regression for each of the independent variables in the model (Hair et al. 2006). No violations were found. Also, homoscedasticity was examined by analyzing the same standardized predicted value vs. studentized residuals as for linearity (Hair et al. 2006); no violations were identified. Finally,

independence of observations was fulfilled in the proper design of the study (Osborn and Waters 2002). As discussed in section 4.2.2.2, data points which had dependency issues were removed from the sample data.

For this study, multiple linear regression analysis required two regression equations (models):

$$VUL\_SEV = \beta_0 + \beta_1 LANGUAGE + \beta_2 SOURCE\_DUMMY1 + \beta_3 SOURCE\_DUMMY2 + \beta_4 SOFT\_TYPE + \beta_5 COMP + \beta_6 TRIAL + \beta_7 PRICE + \beta_8 AUDIENCE + \varepsilon$$

$$VUL\_FRQ = \beta_0 + \beta_1 LANGUAGE + \beta_2 SOURCE\_DUMMY1 + \beta_3 SOURCE\_DUMMY2 + \beta_4 SOFT\_TYPE + \beta_5 COMP + \beta_6 TRIAL + \beta_7 PRICE + \beta_8 AUDIENCE + \varepsilon$$

Where: *VUL_SEV* = Vulnerability severity

VUL_FRQ = Vulnerability frequency

LANGUAGE = Software programming language

SOURCE_DUMMY1, SOURCE_DUMMY2 = Public source code availability

SOFT_TYPE = Software type

COMP = Number of compatible operating systems

TRIAL = Software trial version availability

PRICE = Software Price

AUDIENCE = Software target audience

ε = random error term

4.2.4.2 Assumptions of Poisson Regression:

In addition to multicollinearity and independence assumptions tested before, an important assumption of the Poisson regression model is that the variance has to be equal to the mean. Violation of this assumption does not produce biased coefficients, but it can lead to inefficient parameter estimates and inconsistent standard errors (King 1989). An examination of the data showed that the variance is equal to the mean (mean = 2.044, variance = 2.270). Independence of observations was also fulfilled in the proper design of a study.

The following equation was used for Poisson regression analysis:

$$\log(\text{DIV\_VUL}) = \beta_0 + \beta_1 LANGUAGE + \beta_2 SOURCE\_DUMMY1 + \beta_3 SOURCE\_DUMMY2 + \beta_4 SOFT\_TYPE + \beta_5 COMP + \beta_6 TRIAL + \beta_7 PRICE + \beta_8 AUDIENCE + \varepsilon$$

Where:

DIV_VUL = Diversity of vulnerability types

LANGUAGE = Software programming language

SOURCE_DUMMY1, SOURCE_DUMMY2 = Public source code availability

SOFT_TYPE = Software type

COMP = Number of compatible operating systems

TRIAL = Software trial version availability

PRICE = Software Price

AUDIENCE = Software target audience

ε = random error term

4.3 Results

4.3.1 Multiple Linear Regression Model: Vulnerability Severity

Table indicates that 24.7% of the variance in the vulnerability severity is explained by the model. The overall F-test for the model indicates that the multiple regression is statistically significant at $p < 0.05$.

Table 11:
Model summary for Vulnerability Severity

R	R Square	Adjusted R Square	Std. Error of the Estimate
.497[a]	.247	.203	1.937

Table 8 evaluated each of the predictor variables individually with respect to the dependent variable. In assessing the effect of the other predictor variables on vulnerability severity, the results showed that software type and the number of compatible operating systems were the only predictor variables that had significant effects on vulnerability severity at a 5% significance level. The β coefficients tell us that software type (β =-0.791) and the number of compatible operating systems (β =0.232) had the strongest contributions in explaining vulnerability severity. It's important to note that including and excluding outliers made no difference to the results.

Table 8:
Coefficients for the dependent variable: Vulnerability Severity

	Unstandardized Coefficients		Standardized Coefficients		
	B	Std. Error	Beta	t	Sig.

(Constant)	6.570	.459		14.328	.000*
Software Programming Language	-.165	.172	-.075	-.964	.336
Public Source Code Availability Dummy 1	.082	.198	.031	.414	.680
Public Source Code Availability Dummy 2	-.196	.386	-.036	-.507	.613
Software Type (Web Applications = 1)	-.791	.203	-.301	-3.903	.000*
Number of Compatible Operating Systems	.232	.074	.224	3.124	.002*
Software Trial Version Availability	.118	.185	.044	.641	.522
Software Price	-5.975E-6	.000	-.005	-.066	.947
Software Target Audience	.026	.183	.010	.144	.886

* $p < 0.05$

The final multiple linear regression equation for vulnerability severity model is:

$$VUL\_SEV = 6.570 - 0.791(SOFT\_TYPE) + 0.232(COMP)$$

The equation can be described as follows. First, in relation to web-based software, the severity level of a vulnerability decreases by -0.791 when dealing with web applications. Also, as the number of compatible operating systems increases by 1, the severity level of a vulnerability increases by 0.232

4.3.2 Multiple Linear Regression Model: Vulnerability Frequency

The model, in Table , explained 14.7% of the variance in the dependent variable (vulnerability frequency). The overall F-test for the model indicates that the multiple regression is statistically significant at $p < 0.05$.

Table 9:
Model summary for Vulnerability Frequency

R	R Square	Adjusted R Square	Std. Error of the Estimate
.384[a]	.147	.113	1.235

Table 10 evaluated each of the predictor variables individually with respect to the dependent variable. In assessing the effect of the other predictor variables on vulnerability frequency, the results showed that software type and the number of compatible operating systems were the only predictor variables that had significant effects on vulnerability frequency at a 5% significant level. The β coefficients tell us that software type (β =-0.085) and the number of compatible operating systems (β =0.041) had the strongest contributions in explaining vulnerability frequency. It's important to note that including and excluding outliers made no difference to the results.

Table 10:
Coefficients for the dependent variable: Vulnerability Frequency

	Unstandardized Coefficients		Standardized Coefficients		
	B	Std. Error	Beta	t	Sig.
(Constant)	.608	.085		7.149	.000*
Software Programming Language	.059	.037	.137	1.593	.113
Public Source Code Availability Dummy 1	-.015	.037	-.032	-.411	.681
Public Source Code Availability Dummy 2	-.021	.071	-.021	-.293	.770
Software Type (Web Applications = 1)	-.085	.038	-.176	-2.255	.025*
Number of Compatible Operating Systems	.041	.014	.214	2.959	.003*
Software Trial Version Availability	.037	.034	.078	1.095	.275
Software Price	-1.253E-5	.000	-.052	-.752	.453
Software Target Audience	-.035	.034	-.069	-1.029	.305

* $p < 0.05$

Multiple linear regression for the vulnerability frequency model produced the following equation:

$$VUL\_FRQ = 0.608 – 0.085(SOFT\_TYPE) + 0.041(COMP)$$

The equation can be described as follows. First, in relation to web-based software, the frequency of a vulnerability decreases by -0.085 per month when dealing with web applications. Also, as the number of compatible operating systems increases by 1, the frequency of a vulnerability increases by 0.041 per month.

4.3.3 Poisson Regression Model: Diversity of Vulnerability Types

As seen in Table 11, since the goodness-of-fit chi-squared test is not statistically significant (p = 1.122), this indicates that the model fits reasonably with the data. Table 12 tests the model as a whole by looking if all of the estimated coefficients are equal to zero. However, since the p-value 0.7143 is greater than 0.05, this indicates that the model is not statistically significant and therefore the predictor variables individually are not useful. In assessing the effect of the independent variables on diversity of vulnerability types, the results showed that none of the independent variables had a significant effect on diversity of vulnerability types at a 5% significant level.

Table 11:
Goodness-of-fit for Diversity of Vulnerability Types

	Value	df	Value/df
Deviance	141.214	138	1.023
Pearson Chi-Square	154.832	138	1.122

* $p < 0.05$

Table 12:
Omnibus Test of Vulnerability Types

Likelihood Ratio Chi-Square	df	Sig.
60.983	68	.7143

* $p < 0.05$

4.4 Hypotheses Results

This section presents the overall results of data analysis for the hypotheses outlined in Chapter 2. The hypotheses were tested using multiple linear regression and Poisson regression analysis. Hypotheses relating to the diversity of vulnerability types model are not included since the model was not significant. Also hypotheses related to operating systems (4a1, 4b1, 4c1, 4d2, 4e2, 4f2) and software license (H2a, H2b, H2c) were retracted since the sample data did not include operating systems for issues with data dependency and since software license was removed due to the issue of multicollinearity. Hypotheses results of vulnerability severity and frequency models are summarized below:

H1a, H1b Result Summary:

Hypothesis 1a:

H_0: There is no association between vulnerability severity and software programming language.

H_a: There is an association between vulnerability severity and software programming language.

Hypothesis 1b:

H_0: There is no association between vulnerability frequency and software programming language.

H_a: There is an association between vulnerability frequency and software programming language.

These hypotheses proposed that there is no association between the dependent variables and software programming language.

Table 8 and Table 10 and present the test results of the data analysis. For vulnerability severity, the p-value of software programming language was .336; for vulnerability frequency the p-value was .113. Since the p-value exceeded the significance level α=0.05; therefore we fail to reject the null hypotheses and it was concluded that there is no evidence of an association between each of vulnerability severity and software programming language and between vulnerability frequency and software programming language.

H3a, H3b Result Summary:

Hypothesis 3a:

H_0: There is no association between vulnerability severity and public source code availability.

H_a: There is a positive association between vulnerability severity and public source code availability.

Hypothesis 3b:

H_0: There is no association between vulnerability frequency and public source code availability.

H_a: There is a positive association between vulnerability frequency and public source code availability.

These hypotheses proposed that there is no association between the dependent variables and public source code availability.

Table 8 and Table 10 show the test results. For vulnerability severity, the p-values of public source code availability dummy 1 and dummy 2 were .680 and .613; for vulnerability frequency the p-values of public source code availability dummy 1 and dummy 2 were .681 and .770. In

these analyses, the p-values exceeded the significance level α=0.05; therefore we fail to reject the null hypotheses and it was concluded that there is no evidence of an association between vulnerability severity and public source code availability and between vulnerability frequency and public source code availability.

H4a2, H4b2 Result Summary:

Hypothesis 4a2:

H_0: Vulnerability severity of web-related software is less than or equal to web applications.

H_a: Vulnerability severity of web-related software is greater than web applications.

Hypothesis 4b2:

H_0: Vulnerability frequency of web-related software is less than or equal to web applications.

H_a: Vulnerability frequency of web-related software is greater than web applications.

These hypotheses proposed that vulnerability severity and vulnerability frequency are greater for web-related software than for web applications.

Table 8 and Table 10 show the test results. As interpreted in sections 4.3.1 and 4.3.2, the results confirmed both hypotheses and showed how severity and frequency of vulnerabilities are greater for web-related software when compared with web applications.

H5a, H5b Result Summary:

Hypothesis 5a:

H_0: There is no association between vulnerability severity and the number of compatible operating systems.

H_a: There is a positive association between vulnerability severity and the number of compatible operating systems.

Hypothesis 5b:

H_0: There is no association between vulnerability frequency and the number of compatible operating systems.

H_a: There is a positive association between vulnerability frequency and the number of compatible operating systems.

These hypotheses proposed that there was a positive association between the dependent variables and the number of compatible operating systems. Each of

Table 8 and Table 10 shows the test results. For vulnerability severity, the p-value of the number of compatible operating systems was .002; for vulnerability frequency the p-value was .003. Since the p-values were less than the significance level $\alpha=0.05$; therefore there was sufficient evidence to reject the null hypotheses and it was concluded that there is a positive association between vulnerability severity and the number of compatible operating systems and between vulnerability frequency and the number of compatible operating systems availability.

H6a, H6b Result Summary:

Hypothesis 6a:

H_0: There is no association between vulnerability severity and software trial version availability.

H_a: There is a positive association between vulnerability severity and software trial version availability.

Hypothesis 6b:

H_0: There is no association between vulnerability frequency and software trial version availability.

H_a: There is a positive association between vulnerability frequency and software trial version availability.

These hypotheses proposed that there was a positive association between the dependent variables and software trial version availability. Each of

Table 8 and Table 10 shows the test results. For vulnerability severity, the p-value of software trial version availability was .522; for vulnerability frequency the p-value was .275. Since the p-values were greater than the significance level α=0.05; therefore we fail to reject the null hypotheses and it was concluded that there is no evidence of an association between each of vulnerability severity and software trial version availability and between vulnerability frequency and software trial version availability.

H7a, H7b Result Summary:

Hypothesis 7a:

H_0: There is no association between vulnerability severity and software price.

H_a: There is an association between vulnerability severity and software price.

Hypothesis 7b:

H_0: There is no association between vulnerability frequency and software price.

H_a: There is an association between vulnerability frequency and software price.

These hypotheses proposed that there was no statistically significant association between the dependent variables and software price. Each of

Table 8 and Table 10 shows the test results. For vulnerability severity, the p-value of software price was .947; for vulnerability frequency the p-value was .453. Since the p-values were greater than the significance level $\alpha=0.05$; therefore we fail to reject the null hypotheses and it was concluded that there is no evidence of an association between each of vulnerability severity and software price and between vulnerability frequency and software price.

H8a, H8b Result Summary:

Hypothesis 8a:

H_0: There is no association between vulnerability severity and software target audience.

H_a: There is an association between vulnerability severity and software target audience.

Hypothesis 8b:

H_0: There is no association between vulnerability frequency and software target audience.

H_a: There is an association between vulnerability frequency and software target audience.

These hypotheses proposed that there was no statistically significant association between the dependent variables and software target audience. Each of

Table 8 and Table 10 shows the test results. For vulnerability severity, the p-value of software price was .886; for vulnerability frequency the p-value was .305. Since the p-values were greater than the significance level α=0.05; therefore we fail to reject the null hypotheses and it was concluded that there is no evidence of an association between each of vulnerability severity and software target audience and between vulnerability frequency and software target audience. Table 16 and Table 17 below present a summary of the results of hypotheses relating to vulnerability severity and vulnerability frequency models.

Table 13:
Hypotheses Results of Vulnerability Severity Model:

Hypothesis	**Independent Variable**	**Hypothesis Support**	**Confirmed Prediction**
H1a	Software Programming Language	Not Supported	Yes
H3a	Public Source Code Availability	Not Supported	No
H4a2	Software Type	Supported	Yes
H5a	Number of Compatible Operating Systems	Supported	Yes
H6a	Software Trial Version Availability	Not Supported	No
H7a	Software Price	Not Supported	Yes
H8a	Software Target Audience	Not Supported	Yes

Table 14:
Hypotheses Results of Vulnerability Frequency Model:

Hypothesis	Independent Variable	Hypothesis Support	Confirmed Prediction
H1b	Software Programming Language	Not Supported	Yes
H3b	Public Source Code Availability	Not Supported	No
H4b2	Software Type	Supported	Yes
H5b	Number of Compatible Operating Systems	Supported	Yes
H6b	Software Trial Version Availability	Not Supported	No
H7b	Software Price	Not Supported	Yes
H8b	Software Target Audience	Not Supported	Yes

Chapter 5 Results and Conclusion

The purpose of this research was to study software characteristics and to analyze how they predict vulnerability risks. The objective was to examine this matter from both researchers' and practitioners' point of views in order to have a better understanding of vulnerability risks and how they can be predicted in the real world using software characteristics. Although there is a growing awareness of the importance of software security and vulnerabilities, little research has actually examined how to predict vulnerability risks. This work creates a novel model and presents a quantitative measure for predicting vulnerability risks.

This chapter discusses the findings and conclusions of this research study. Conclusions and supporting evidence are summarized and the strength of the evidence with respect to the results is explained. Finally, this chapter talks about the results to provide both research and practical implications for this research. This chapter concludes with limitations and suggestions for future research.

5.1 Vulnerability Survey Conclusion

Prior studies have researched vulnerabilities based on collecting vulnerability data exclusively without taking into consideration the importance of opinions of IT practitioners (Alhazmi and Malaiya 2005a 2005b 2006, Gopalakrishna and Spafford 2005, and Woo et al. 2006). It was important in this study to consider and evaluate the opinions of IT practitioners regarding vulnerability risks. IT practitioners comprise the majority of software developers and adopters and they are considered a key element when it comes to evaluating software and dealing with

vulnerabilities. Therefore, it's essential to understand how IT practitioners predict vulnerability risks of software.

Based on the survey results, it was apparent that the majority of respondents considered software type to have the most effect on vulnerability risks. Survey results implied uniformity of opinions as the agreement rate was 77.54% among 187 respondents from 19 different industries. Given the vast amount of software types and their different exposure levels to security threats, it's no surprise that IT practitioners believe that software type has an effect on vulnerability risks (Lloyd 2003). Certain types of software products such as web-based software and web applications are exposed not only to threats from within the network but also to threats from outside the network (SANS 2011). For instance, with respect to SQL injection, web-based software are more vulnerable to this kind of attack than other software types (ex: utility software) since web-based software are exposed to network attacks while the latter are not.

Furthermore, with a 73.80% agreement rate among all respondents, the survey showed that the number of compatible operating systems was the second variable deemed to have significant effects on vulnerability risks. This finding implies that IT practitioners are aware of the association between software compatibility and vulnerability risks. As discussed in Chapter 2, software products inherit certain aspects of the operating systems they are running on including weaknesses, vulnerabilities and threats. Therefore, for IT practitioners, dealing with a product which is compatible with many operating systems means increasing vulnerability risks and security breaches.

Of the remaining variables, and as expected, the survey illustrated unanimous disagreement towards any association between vulnerability risks and each of (software programming

language, software trial version availability, software price and software target audience). However, the only variable that seemed to be controversial among the respondents was public source code availability. Eighty five respondents (45.54%) agreed that public source code availability had an effect on vulnerability risks while 76 respondents (40.64%) disagreed, and 26 respondents (13.90%) were neutral. This finding confirms the long debated subject regarding security of open source (Brown 2002) and it verifies the existing major differences in opinions among IT practitioners. It's my belief that further details and more in-depth research is necessary to collect data regarding the controversy behind public source code availability and vulnerability risks.

And last, when respondents were asked to rank the top three software characteristics which have the most effect on vulnerability risks, they ranked each of software type, number of compatible operating systems and public open source availability. The ranking illustrates the amount of effect each characteristics has on vulnerability risks from the viewpoint of the respondents.

5.2 Vulnerability Data Analysis Conclusion

This paper extends previous research (Alhazmi and Malaiya 2005a 2005b 2006, Gopalakrishna and Spafford 2005, and Woo et al. 2006) by examining multiple aspects of vulnerability risks including: severity level, frequency of occurrence and diversity of types. Three novel vulnerability prediction models were proposed and explored for their applicability of using software characteristics to predict vulnerability risks. Among the three proposed models, two models (vulnerability severity and vulnerability frequency) were found to have significant associations with software characteristics. The third model, namely diversity of vulnerability types did not show any significant association with software characteristics.

The vulnerability severity model was fitted to vulnerability severity data obtained from NVD and the fit was found to be statistically significant. This model observed each of software type and the number of compatible operating systems to have significant positive association with vulnerability severity. More specifically, it was found that web-related software is susceptible to more severe vulnerabilities than web applications. This finding confirms with the fact that, over time, hackers shift focus to target different software types (Schmidt et al. 2009, Leavitt 2005). Also, the results showed a positive association between the number of compatible operating systems and the severity of vulnerabilities. This result confirms Krsul (1998) and Wurster (2010) findings that software inherit vulnerabilities from compatible operating systems. Furthermore, based on the vulnerability severity model, this research hypothesized a positive association between vulnerability severity and each of (public source code availability and software trial version availability); however the results were not statistically significant.

The second proposed model was based on vulnerability frequency. The model was fitted to vulnerability frequency data and the fit was found to be statistically significant. This model also observed each of software type and the number of compatible operating systems to have significant positive association with vulnerability frequency. Further analysis revealed that web-related software is susceptible to more frequent vulnerabilities than web applications. These results confirm with recent security reports which showed the increasing frequency of attacks and vulnerability exploitations on web-related software (SANS 2010). Also, the results showed a positive association between the number of compatible operating systems and the severity of vulnerabilities.

Furthermore, the hypothesized positive association between vulnerability frequency and each of

(public source code availability and software free version availability) was not found to be statistically significant.

There are numerous potential reasons for why the positive associations between the independent variables (public source code availability and software trial version availability) and the dependent variables were not statistically significant in both models:

First, while the hypothesized positive association between vulnerability severity and public source code availability and between vulnerability frequency and source code availability wasn't significant, it's likely that no association is present (Payne 2002). Another reason for this finding is the contrary argument that making source code publically available makes software more secure (Hoepman and Jacobs 2007). But despite such propositions, the results did not show any positive or negative associations.

Second, the hypothesized positive association between vulnerability severity and software trial version availability and between vulnerability type and software trial version availability was based on the belief of hackers using a trial version of the software to discover vulnerabilities. However, the results showed no association. It's likely that with the wide spread of pirated copies of software products (Business Software Alliance 2008, Banerjee et al. 2005), the availability of a trial version cease to remain useful to hackers since they can have immediate access to the pirated version to find vulnerabilities. It's also possible that with the recent trend in the software market toward free software and with the shift headed for the licensing model and the content-targeted advertisement (Frank 2006), a trial version is no longer being released by vendors.

This research found that software characteristics can't be used to predict diversity of vulnerability types. Besides the possibility that there is no underlying relationship between diversity of vulnerability types and software characteristics, there are other explanations for this finding. First, this can be caused by the classification of vulnerability types in NVD as some vulnerability types are underrepresented due to errors or difficulties in classification (NVD 2010). For example, NVD classifies web parameter tampering as an authentication error where in practice they are normally classified as validation error. Second, through the analysis I found that about 20% of vulnerability types in NVD are classified under 'other' category which further limits the overall diversity of vulnerability types. Third, it's possible that the limited choice of our software categories could have narrowed the diversity of vulnerability types. A more diverse set of vulnerability types might have been attained by extending software types to include other categories.

5.3 Reconciliation of Subjective and Objective Results

This research demonstrated a link between subjective and objective results. It's of great significance to note that the level of agreement between subjective and objective results was particularly high as both survey respondents and data analysis found each of software type and the number of compatible operating systems to be important for predicting vulnerability risks. This overlap demonstrates a substantial level of validity, and it stresses the significance of the practicality of this research. Moreover, both subjective and objective results found no association between vulnerability risks and each of (software programming language, software trial version availability, software price and software target audience). A slight discrepancy between subjective and objective results was public source availability. While secondary data analysis

found no association between the dependent variables and public source availability, primary respondents on the other hand ranked public source availability among the top three characteristics affecting vulnerability risks. This discrepancy can be related to the fact the IT practitioners remain skeptical about the security of open source software (Forrester Research 2007).

5.4 Research Implication

In terms of implications for academic research, our study offers new insights into the area of vulnerability prediction and software security. In this research, the focus was more on the external aspects of software products than internal ones. Predicting vulnerabilities based on this approach has not been paid enough attention especially in the software security field (Alhazmi and Malaiya 2006). This paper tackles this limitation in research and explores the importance of external software characteristics and their effects on vulnerabilities. A research implication is the discovery of the significant role of software characteristics in predicting vulnerability risks. More specifically, the results demonstrated that software type and the number of compatible operating systems have significant effects on vulnerability risks. More importantly, this research provided specific information on which type of software products is more susceptible to vulnerabilities. The other research implication is the creation of two vulnerability prediction models. Such models answered the call of Alhazmi and Malaiya (2006) for the need of more quantitative vulnerability predictive models. Another research implication is the consideration of IT practitioners' opinions in regards to vulnerability risks. This is relatively a new contribution of knowledge to the area of software security and it enables researchers to understand vulnerability risks from new perspectives. Finally, an obvious research implication of the obtained result is the

necessity of extending vulnerability risks and software characteristics to encompass other variables in order to improve predictability of vulnerability risks.

5.5 Practical Implication

This study has several practical implications for IT practitioners and software vendors who wish to develop/adopt a secure software. By identifying software characteristics that can affect vulnerability risks, software vendors and IT practitioners who are keen on security can use them as guidelines for predicting the severity level and frequency of future vulnerabilities. Moreover, software vendors and IT companies may wish to establish a security guideline based on the results for developing/adopting software. Such guidelines can establish security priorities with regards to the type of the software and the number of compatible operating systems. For example, by having knowledge of which software type is more susceptible to vulnerabilities, software developers can put more effort into writing secure code and software testers can devote more time into finding bugs and flaws. Additionally, adopters of software which is predicted to have more severe and frequent vulnerabilities can better equip their security team to monitor for any breaches.

5.6 Limitations

The results presented in this study must be considered in view of certain limitations. First, the data used in the analysis comes from the National Vulnerability Database (NVD). NVD is known as the most reliable vulnerability database but it still suffers from inaccuracies and missing information. In future research, it would be more appropriate to combine NVD with other vulnerability databases such as Open Source Vulnerability database (OSVD) to overcome

this limitation. Second, it's important to note that this research analyzed publically reported vulnerabilities without considering unreported/undisclosed data. While this drawback is common among vulnerability research, overcoming this limitation can be achieved through direct contact with software vendors which can be an extension for future research. Third, this was a cross-sectional study; therefore we suggest prospective studies to assess the effects of vulnerability risks through longitudinal data. Fourth, this research has attempted to build a prediction model based on diversity of vulnerability types; however the model wasn't statistically significant. As discussed before, I believe that the classification system of vulnerability types used in NVD is not detailed enough, therefore for future studies, a better classification system should be used to overcome this limitation. Finally, the sample data consisted of two software types only. As discussed previously, this limitation may have had an effect on the predictability level of the models.

5.7 Future Research

Future research should continue to employ precise hypotheses that include other software characteristics in order to better predict and understand their effect on vulnerability risks. Additional refinements to hypotheses include, considering sub-types of software categories, focusing on more specific paradigms of software programming languages or on the software development model where vulnerability risks may be most evident. Researchers may also wish to develop a theoretical understanding of how vulnerability types are classified in order to build a better model for diversity of vulnerability types. Further work is also needed to improve the prediction capabilities of the proposed models as this research relies entirely on external software characteristics. Future research can combine both internal and external characteristics to produce

a better prediction model. Further analysis of the vulnerability risks is also needed in order to build a more comprehensive model with better predictabilities. Vulnerability risks such as complexity/sophistication level could be explored to answer questions such as: is there an association between complexity of vulnerabilities and software characteristics? Additionally, the disclosure aspect of vulnerabilities needs to be analyzed as there are several mediums to publish vulnerabilities such as: official disclosure (ex: NVD) and unofficial disclosure (ex: mailing-lists, discussion forums…etc). Recognize how vulnerabilities are disclosed can significantly increase our understanding of unknown relationships between vulnerability disclosure and vulnerability risks. Furthermore, since improving predictability of the proposed models is a central objective, future research can explore the possibility of merging the proposed models in this paper with existing ones such as the Vulnerability Discovery Model (VDM) built by Alhazmi and Malaiya (2005a).

References

1. Alex-Foss, J. and Barbosa, S. (1995): Assessing computer security vulnerability, ACM SIGOPS Operating Systems Review, 29(3).

2. Alhazmi OH, Malaiya YK. Quantitative vulnerability assessment of systems software. In: Proceedings of 51st annual reliability and maintainability symposium, Alexandria, VA; January 2005a. p. 615–20.

3. Alhazmi O. H. and Malaiya Y. K., "Modeling the Vulnerability Discovery Process," Proc. Int. Symp. Software Reliability Eng, Nov. 2005b, pp. 129-138.

4. Alhazmi O. H. and Malaiya Y. K., “Measuring and enhancing predic- tion capabilities of vulnerabilities discovery models for Apache and IIS HTTP servers,” in Proc. 17th IEEE International Symposium on Soft- ware Reliability Engineering (ISSRE’06), 2006a, pp. 343–352.

5. Alhazmi, O.H. Malaiya Y.K., "Prediction capabilities of vulnerability discovery models," rams, pp.86-91, RAMS '06. Annual Reliability and Maintainability Symposium, 2006b.

6. Alhazmi, OH., Y. K. Malaiya , I. Ray, " Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems," Computers and Security Journal, Volume 26, Issue 3, May 2007, Pages 219-228.

7. Alhazmi, O. H., and Malaiya Y. K. (2008). Application of vulnerability discovery models to major operating systems. IEEE Transactions on Reliability, vol. 57, no. 1, pp. 14-22.

8. Anderson, R. J. (2002). Security in open versus closed systems—The dance of Boltzmann, Coase and Moore, in Open Source Software. Economics, Law and Policy. Toulouse, France, pp. 20–21.

9. Anupam, V. and Mayer, A. (1998). Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies, In Proceedings of the 7th USENIX Security Symposium, 187-200.

10. Arbaugh, W.A., Fithen, W. L. and McHugh, J. (2000), Windows of Vulnerability: A Case Study Analysis, IEEE Computer.

11. Avizienis A., Laprie J.C., Randell B. & Landwehr C. (2004) Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable and Secure Computing 1,1 (2004) 11- 33.

12. Banerjee, D., Khalid, A. M., and J.-E. Sturm (2005). Socio-economic Development and Software Piracy. An Empirical Assessment. Applied Economics 37, p2091-2097.

13. Bishop, M., Bailey, D. (1996). A Critical Analysis of Vulnerability Taxonomies. Technical Report CSE-96-11, Department of Computer Science at the University of California at Davis.

14. Brown, W. A., and Booch, G. (2002). Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors, in C. Gacek (ed.), Software Reuse: Methods, Techniques, and Tools, New York: Springer-Verlag, 123-13.

15. Browne, H., McHugh, J., Arbaugh, W., and Fithen, W. (2001). A Trend Analysis of Exploitations. IEEE Symposium on Security and Privacy.

16. Business Software Alliance. (2008). Sixth Annual BSAIDC Global Software Piracy Study. Retrieved January 31st 2012 from: http://global.bsa.org/globalpiracy2008/studies/globalpiracy2008.pdf

17. Chellappa, R. K., Shivendu, S. (2005). Managing piracy: pricing and sampling strategies for digital experience goods in vertically segmented markets. Information Systems Research, 16(4), p 400-417.

18. Chin, W. W. (1998). The partial least squares approach to structural equation modeling. Modern Methods for Business Research, 295, 336.

19. Comino, S., Manenti, F. M. and Parisi, M. L. (2005). From planning to mature: On the determinants of open source take off. Retrieved on Nov 16th 2010 at http://opensource.mit.edu

20. Cowan, C., Wagle, P., and Pu, C. (1999). Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade, DARPA Information Survivability Conference and Expo.

21. CWE . (2010). Common Weakness Enumeration. Retrieved on Sept 16th 2010 at http://cwe.mitre.org

22. CVE. (2010). Common Vulnerability and Exposures Retrieved on Nov 6th 2010 at http://cve.mitre.org/

23. Dacier, M., Deswarte Y., and Kaâniche M. (1996). Quantitative Assessment of Operational Security: Models and Tools Technical Report 96493, Laboratory for Analysis and Architecture of Systems.

24. De *Ru*, W.G., Eloff, J.H.P. (1996), "*Risk analysis modelling with the use of fuzzy logic*", Computers and Security, Vol. 15 No.3, pp.239-48

25. Dillman, D. A. (2000). Mail and Internet Surveys: The Tailored Design Method, Wiley, New York.

26. Dr Dobbs Journal (2009). Open Source Study Reveals High Level of Code Reuse. Retrieved on Jan 25th 2011 at http://www.drdobbs.com/open-source/216401796

27. Ernst and Young (2007). 10th Annual Global Information Security Survey: Achieving a Balance of Risk and Performance.

28. Eusgeld, I., Kroger, W., Sansavini, G., Schläpfer, M., & Zio, E. (2009). The role of network theory and object-oriented modeling within a framework for the vulnerability

analysis of critical infrastructures. Reliability Engineering & System Safety, 94(5), 954–963.

29. Festa, P. (2001). Empire building, Mozilla style. Retrieved on January 2nd 2011 at http://news.cnet.com/2100-1023-252083.html

30. Forrester Research. (2007). Enterprise and SMB Software Survey, North America and Europe Q3.

31. Gefen, D., Straub, D. W. and Boudreau, M. C. (2000). Structural equation modeling and regression: Guidelines for research practice. Comm. AIS, 4(7), p1–78.

32. Geer, D. (2009). The OS Faces a Brave New World. Computer, IEEE press. 42(10), 15–17.

33. Geer, D. (2007). Addressing the Nation's Cybersecurity Challenges: Reducing Vulnerabilities Requires Strategic Investment and Immediate Action. Cybersecurity, and Science and Technology.

34. Gomez J and Dasgupta D, "Evolving Fuzzy Classifiers for Intrusion Detection", Proceedings of the IEEE, 2002.

35. Gopal, R., and Sanders, G. (2000). Global software piracy: You can't get blood out of a turnip. Communications of the ACM, 43(9), 83–89.

36. Gopalakrishna R. and Spafford E. H. A trend analysis of vulnerabilities. Technical Report 2005-05, CERIAS, Purdue University, May 2005.

37. Grimm, L. G. and Yarnold, P. R. (1995). Reading and Understanding Multivariate Statistics. Washington D.C. American Psychological Association.

38. Hair, J.F., Anderson, R.E., Tatham, R.L. and Black, W.C. (1998). Multivariate Data Analysis
5th ed., Prentice-Hall, Upper Saddle River, NJ.

39. Heiman, A., E. Muller (1996). Using demonstration to increase new product acceptance: controlling demonstration time. Journal of Market Research. 33(4), 422-430.

40. Hoare, C. A. R. (1974). Hints on Programming Language Design. Computer Systems Reliability: State of the Art Report, Vol. 20, pp. 505-34.

41. Hoffer, J. A., George, J. F., and Valacich, J. S. (2002).Modern Systems Analysis & Design (Third ed.). New Jersey: Prentice Hall.

42. Hoglund, G., McGraw, G. (2004). Exploiting Software: How to Break Code. Boston: Addison-Wesley.

43. Hoepman, J. and Jacobs, B. (2007). Increased Security Through Open Source. Communications of the ACM, pp. 79-83.

44. Joh H. and Malaiya Y. K., "Seasonal Variation in the Vulnerability Discovery Process, " Proc. 2nd IEEE Int. Conf. Software Testing, Verification, and Validation, April 2009, pp. 191-200.

45. Jonsson, E. and Olovsson, T. (1997). A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. IEEE Transactions on Software Engineering, 23(4).

46. Kim J., Y. K. Malaiya and I. Ray, "Vulnerability Discovery in Multi-Version Software Systems," Proc. 10th IEEE Int. Symp. on High Assurance System Engineering (HASE), Dallas, Nov. 2007, pp. 141-148.

47. Kotadia, M. (2005). Samy opens new front in worm war. Retrieved on January 2nd 2011 at http://news.cnet.com/Samy-opens-new-front-in-worm-war/2100-7349_3-5897099.html?tag=mncol

48. Krsul I. V. (1998). Software Vulnerability Analysis. PhD thesis, Purdue University.

49. Kumar S. and Spafford E. , "A Pattern Matching Model for Misuse Intrusion Detection," Proceedings of the 17th National Computer Security Conference, West Lafayette, 1994.

50. Lloyd, S. J. (2003). Practicing Software Engineering in the 21st Century. IRM Press.

51. Landwehr, C., Bull, A., McDermott, J. and Choi, W. (1994) A taxonomy of computer program security flaws, ACM Computing Surveys, 26, 3, 211-254.

52. Leavitt, N. (2005). Instant Messaging: A New Target for Hackers (2005). IEEE Computer, 38, 7.

53. Lee, S. C., and Davis, L. B. (2003). Learning from experience: Operating system vulnerability trends. IT Professional, 5(1).

54. Lee, Y. J. and Tan, Y. (2007). An empirical study of software sampling: Categorical Heterogeneity and Vendor Strategy. Proceedings of Workshop on Information Technology and Systems. p73-78.

55. Lerner, J., and Tirole, J. (2005). The economics of technology sharing: Open source and beyond, Journal of Economic Perspectives, 19(2), pp. 99–120.

56. Lindqvist U. and E. Jonsson. 1997. How to systematically classify computer security intrusions. In Proceedings of the 1997 IEEE Symposium on Security and Privacy held in Oakland, CA.

57. Littlewood, B., Brocklehurst, S., Fenton, N., Mellor, P., Page, S. and Wright, D. (1993). Towards Operational Measures of Computer Security, Journal of Computer Security, Volume (2) 2/3. pp 211-230.

58. Lorek, K. S., C. L. McDonald, and D. H. Patz, "A Comparative Examination of Management Forecasts and Box-Jenkins Forecasts of Earnings", The Accounting Review, (1976) 321-330.

59. Lowis, L., Accorsi, R.: Vulnerability analysis in SOA-based business processes. IEEE Transactions on Services Computing (2010).

60. Luftman, J. and McLean, E. R. (2004). Key Issues for IT Executives, MIS Quarterly Executive, pp89-104.

61. Manadhata, P., Wing, J.M. (2005). An attack surface metric. Technical Report CMU-CS-05-155.

62. Mansfield-Devine, S. (2008). Danger in the clouds. Network Security, (12), 9-11

63. Martin, R. A. and Barnum, S. (2008). A status update: The common weaknesses enumeration. Technical report, MITRE Corporation. Retrieved on February 3rd 2011 at http://cwe.mitre.org/documents/cwe_update.pdf

64. Martin, R. A., Christey, S., Jarzombek, J., "The Case for Common Flaw Enumeration". "NIST Workshop on Software Security Assurance Tools, Techniques, and Metrics", November, 2005 Long Beach, CA.

65. McGraw, G., Software Security: Building Security In. Boston, NY: Addison-Wesley, 2006.

66. McGraw, G. (2006). Software Security: Building Security In. Boston, NY: Addison-Wesley.

67. Meunier P. (2007). Classes of vulnerabilities and attacks. Handbook of Science and Technology for Homeland Security: Wiley

68. Mell P., Scarfone K., and Romanosky S., "A Complete Guide to the Common Vulnerability Scoring System Version 2.0," Forum of Incident Response and Security Teams, June 2007, http://www.first.org/cvss/cvss-guide.html.

69. Mercuri, R. T. (2003). Analyzing security costs. Comm. ACM 46(6) 15–18

70. Microsoft. (2010). Microsoft Security Response Center Security Bulletin Severity Rating System (Revised, November 2002). Retrieved Sept 8th 2010 at: http://www.microsoft.com/technet/security/bulletin/rating.mspx

71. Moore, A. P.; Ellison, R. J.; & Linger, R. C. Attack Modeling for Information Security and Survivability (CMU/SEI-2001-TN-001, ADA388771). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

72. Ortalo, R., Deswarte, Y., Kaâniche, M. (1999). Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security, IEEE Transactions on Software Engineering 25,5, p.633-650.

73. Ozment, A. "Improving Vulnerability Discovery Models," Proc. 2007 ACM Workshop on Quality of Protection, ACM Press, 2007, pp. 6-11.

74. Parrend, P., Frenot, S. (2008). Classification of Component Vulnerabilities in Java Service Oriented Programming (SOP) Platforms. In: CBSE 2008. LNCS, p80-96, Springer Berlin/Heidelberg.

75. Payne, C. (2002). On the security of open source software. Information Systems Journal, 12(1), 61-78.

76. Peace, A. G., Galletta, D. F., & Thong, J. Y. L. (2003). Software piracy in the workplace: A model and empirical test. Journal of Management Information Systems, 20, 153-177.

77. Perens, B. (2009). How Many Open Source Licenses Do You Need?. Retrieved on December 23rd 2011 at: http://itmanagement.earthweb.com/osrc/article.php/3803101/Bruce-Perens-How-Many-Open-Source-Licenses-Do-You-Need.htm

78. Pham, N. H., Nguyen, T. T., Nguyen, H. A., Wang, X., Nguyen, A. T., and Nguyen, T. N. (2010). Detecting Recurring and Similar Software Vulnerabilities. Proceedings of the 32nd International Conference on Software Engineering (ICSE 2010).

79. Poulsen, K. (2003). Thwarted Linux backdoor hints at smarter hacks. Retrieved on January 31st 2011 at: http://www.securityfocus.com/news/7388

80. Premkumar, G., Ramamurthy, K., Nilakanta, S. (1994), "Implementation of electronic data interchange: an innovation diffusion perspective", Journal of Management Information Systems, Vol. 11 No.2, pp.157-86.

81. Qualys. 2010. Severities Knowledgebase. Retrieved Sept 8th 2010 at: http://www.qualys.com/research/knowledge/severity/

82. Raymond, E.S., and B. Young. (2001). The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly, Sebastopol, CA.

83. Rescorla, E., Is finding security holes a good idea?, Workshop on Economics and Information Security 2004, May 2004

84. Rogers, E. M. (1983). Diffusion of innovations (3rd ed.). New York: Free Press.

85. Rogers, E. M. (1995). Diffusion of innovations (4th ed.). New York: Free Press.

86. Ross ,. A. Security in open versus closed systems the dance of Boltzmann, Coase and Moore. In: Conference on open source software: economics, law and policy, Toulouse, France; June 2002, p. 1–15.

87. Ryan T. (2009). Modern Regression Methods. New York, NY: Wiley Interscience.

88. SANS. (2010). An Overview of Threat and Risk Assessment. Retrieved on Sept 9th 2010 at: http://www.sans.org/reading_room/whitepapers/auditing/overview-threat-risk-assessment_76

89. SANS. (2010). Cyber Security Risks. Retrieved on February 1st 2012 from: http://www.sans.org

90. SANS. (2011). Framework for Network Security Intelligence. Retrieved on February 1st 2012 from: http://www.sans.org

91. Schiffman, M. (2007), A Complete Guide to the Common Vulnerability Scoring System (CVSS). http://www.first.org/cvss/cvss-guide.html

92. Schmidt, A.D., Schmidt, H.G., Batyuk, L., Clausen, J.H., Camtepe, S.A., Al-bayrak, S. (2009). Smartphone Malware Evolution Revisited: Android Next Target? In: Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE).

93. Shapiro, C., and Varian, H. R. (1990). Information rules: a strategic guide to the network economy. Boston, Mass.: Harvard Business School Press, 1999.

94. Shaw, D. S.; Post, J. M. and Ruby, K. G. (1999). Inside the minds of the insider, Security Management, 43(12) 34-44.

95. Shin, Y. and Williams, L., An Empirical Model to Predict Security Vulnerabilities Using Code Complexity Metrics, IEEE Empirical Software Engineering and Metrics (ESEM) 2008 short paper, Kaiserslautern , Germany, pp. 315-317.

96. Spitzner, L. (2002). Honeypots: Tracking Hackers. Boston: Addison-Wesley.

97. Stathopoulos, V., Kotzanikolaou, P., and Magkos E. "Secure Log Management for Privacy Assurance in Electronic Communications", In: Computers & Security, Elsevier, Volume 27 (7-8), pp. 298-308, 2008

98. Sutherland, I., Kalb, E. G., Blyth, A., and Mulley, G. (2006). An empirical examination of the reverse engineering process for binary files. Computers & Security, 25(3), 221–228.

99. Telang, R., and Wattal, S. (2005). Impact of Software Vulnerability Announcements on the Market Value of Software Vendors -- an Empirical Investigation. Proc. Fourth Workshop on the Economics of Information Security, 2005.

100. Tevis, J. (2005). Automatic Detection of Software Security Vulnerabilities in Executable Program Files. Dissertation submitted to the Department of Computer Science. Auburn University. Auburn, AL.

101. Tsipenyuk, K., Chess, B., McGraw, G. (2006) "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors", IEEE Security and Privacy, Vol. 3, No. 6, 81-84.

102. Tornatzky, L. G., and Klein, K. "Innovation Characteristics and Innovation Implementation: A Meta-Analysis of Findings," IEEE Trans- actions on Engineering Management (29:1), 1982, pp. 28-45.

103. Volpano, D. (1997) Provably-secure programming languages for remote evaluation, tech. rep., Computer Science Dept., Naval Postgraduate School, Monterey, CA.

104. Walia, N., Rajagopalan, B. and Jain, H. (2006). Comparative Investigation of Vulnerabilities in Open Source and Proprietary Software – An Exploratory Study. 12th Proceedings of the Americas Conference on Information Systems-AMCIS.

105. Woo S-W., "An Analysis of Vulnerabilities in Web-servers and Browsers Using Time based and Effort-based Models," Thesis, CS Dept, Colorado State University, 2006.

106. Woo, S-W., Alhazmi O.H. and Malaiya Y.K., "An Analysis of the Vulnerability Discovery Process in Web Browsers", Proc. 10th IASTED Int. Conf. on Software Engineering and Applications, Nov. 2006.

107. Wurster, G. (2010). Security Mechanisms and Policy for Mandatory Access Control in Computer Systems. PhD thesis, Carleton University.

108. Zaltman, G., R. Duncan and J. Holbeck (1973). Innovations and Organisations. NewYork: John Wiley & Son.

Appendix A - Survey Questionnaires

Predicting Software Vulnerability Risks Using Software Characteristics

The purpose of this project is to predict software vulnerability risks using software characteristics. We have identified several software characteristics which we believe could affect vulnerability risks. The following survey has questions about those factors.

(Section I)

What is your job title?

How long have you been working at this position?

Rate your security knowledge

What is the industry of your company?

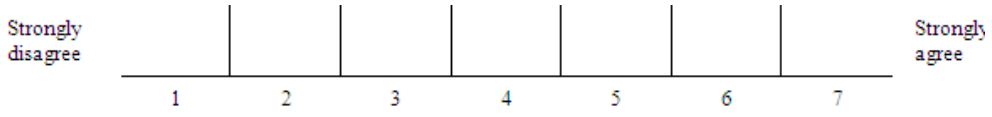
What is the sector of your company?

(Section II)

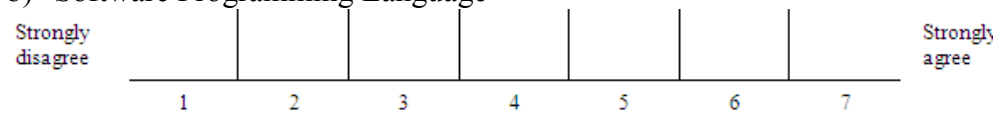
Please rate how much you agree or disagree with the following statements:

Vulnerability risk is affected by:

a) Software Type



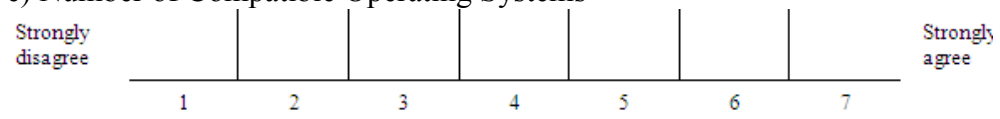
b) Software Programming Language



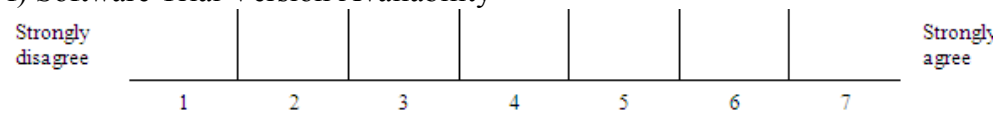
c) Source Code Availability

d) Software License

e) Number of Compatible Operating Systems



f) Software Trial Version Availability



g) Software Price

h) Software Target Audience

(Section III)

Rank the top 3 software characteristics that have the most effect on vulnerability risks (starting with the highest)

Rank # 1

Rank # 2

Rank # 3

Section (IV)

This section provided respondents with the ability to comment regarding any previous responses in Section II except for those marked as 4 “Neutral”.

Section (V)

- We appreciate any comments you may have about the questionnaire.
- Are there any other software characteristics which can help predict vulnerability risk that the survey didn’t include?
- Other comments?