Software Selection based on Quantitative Security Risk Assessment

Ruma Das
Engineering Management and System Engineering
George Washington University
Washington DC, USA

Shahram Sarkani
Engineering Management and System Engineering
George Washington University
Washington DC, USA

Thomas A. Mazzuchi
Engineering Management and System Engineering
George Washington University
Washington DC, USA

ABSTRACT
Multiple software products often exist on the same server and therefore vulnerability in one product might compromise the entire system. It is imperative to perform a security risk assessment during the selection of the candidate software products that become part of a larger system. Having a quantitative security risk assessment model provides an objective criterion for such assessment and comparison between candidate software systems. In this paper, we present a software product evaluation method using such a quantitative security risk assessment model. This method utilizes prior research in quantitative security risk assessment, which is based on empirical data from the National Vulnerability Database (NVD), and compares the security risk levels of the products evaluated. We introduced topic modeling to build a security risk assessment model. The risk model is created using Latent Dirichlet Allocation (LDA) to classify the vulnerabilities into topics, which are then used as the measurement instruments to evaluate the candidate software product. Such a procedure could supplement the existing selection process, to assist the decision makers to evaluate open-source software (OSS) systems, to ensure that it is safe and secure enough to be put into their environment. Finally, the procedure is demonstrated using an experimental case study.

General Terms
Risk Management, Measurement, Security.

Keywords
Software Security, quantitative risk assessment, Software evaluation, Topic Modeling, LDA.

1. INTRODUCTION
The system engineering approach to selecting a secure system ensures that the system selected meets the mission of the organization and the security level needed by the missions [1]. For example, a system selected to hold patient data must be secure enough to protect patient data and provide necessary controls to prevent unauthorized access. As per International Council on Systems Engineering (INCOSE), system engineering is "an interdisciplinary approach and means to enable the realization of successful systems" [2]. For a system to be successful, a security risk assessment of the software product that will eventually become part of a larger system is important. In a software system developed in-house, there is an opportunity to make it secure by taking proactive measures during the entire development life cycle. In many cases, the security assessment is conducted in the development or verification and validation (V&V) phase through vulnerability scanning [3], secure code review, threat modeling, etc. However, when an off-the-shelf or open source software product is evaluated, the system engineers have no control over design and build phase. In this case, a security risk assessment becomes even more essential and must be conducted upfront during the initial evaluation. As pointed out by Schneidewind, an insecure COTS or OSS product may be a weak link in the system environment [4] and impact the reliability of the entire system. In addition to black box testing and operation testing, impact on other systems in the environment must be tested using fault injection testing on the candidate OSS product. Recent research has presented a framework to ensure that security is actively considered during the software development life cycle [5]. However, for an OSS, such an opportunity is not available. Therefore it is more critical that security be evaluated before implementing the OSS in an operational setting.

Security flaws in a software product increase the security risk to the whole environment in which it operates. When evaluating an open or closed source off-the-shelf software system for use, the software security assessment must be conducted to ensure that the product has a relatively low security risk. The term 'software product' in this paper refers to a web-based database-driven application. When comparing multiple candidate software products, an objective measure is needed for such evaluation. The quantitative security risk assessment is gaining momentum in research for decision-making, risk assessment and resource allocation [6-12]. Prior studies in quantitative software security risk assessment have mainly focused on: browsers, application servers, database servers, individually or on entire network topology. A new study is needed to identify how we can apply or extend such a model and metrics to compare and select software products such as web-based applications.

This study introduces a procedure to select from a multitude of OSS products and is based on a quantitative security risk assessment. The relevance of the research can be observed in a case study performed by Austin et al., which evaluated multiple open source electronic health record systems. The researchers detected approximately 1321 issues through static analysis and 710 through dynamic analysis[13]. Security assessment during OSS selection has been proposed in prior works[14]. However, skilled security professionals are rarely used during the initial selection process of a software system. Most security assessments are performed after the system has been implemented. The framework discussed below can be utilized by practitioners to augment the existing security

assessment process during the early stages of software product evaluation.

Our approach investigates the application of topic models for creating the quantitative security risk assessment model. The reason for choosing the approach of having the machine generate a list of topics is that not all vulnerabilities have been categorized in NVD using CWE (Common Weakness Enumeration) groups[15]. As a result, it is difficult to determine the actual likelihood or frequency, and thereby, severity of a vulnerability group or threat. Finally, we selected the Design Science Research Methodology (DSRM) to perform this study. This research methodology selection is based on our goal to propose an artifact to support the evaluation of off-the-shelf or OSS software products before adoption using the quantitative security risk assessment model. The activities in our research have been outlined below, adopting the steps mentioned by Peffers et al. for presenting the information derived through Design Science Research Methodology (DSRM) [37].

The primary contributions of this paper are:

1) Demonstration of usage of topic modeling for vulnerability grouping.
2) Identification of dimension for security assessment of candidate software products
3) Development of a method to enable practitioners to review candidate products objectively
4) Outline of an experiment to perform the procedure

The remainder of the paper is organized as follows: Section 2 provides background information and definitions. Section 3 outlines the prior work on which this research is based. Section 4 provides reasons for research methodology selection. Section 5 describes the research methodology. Section 6 presents results and discussion. Section 7 states limitations and Section 8 concludes the paper.

2. BACKGROUND

This section provides brief background information on the concepts used in this paper. For example, the paper presents 'Topic Modeling' as a method for clustering the data and 'Design Science' as a research methodology. We describe these concepts in this section.

2.1 Topic Modeling

Topic Modeling is used to analyze large corpora of text, in order to categorize them into clusters identified by a set of terms or tags. The term topic modeling refers to the technique of identifying themes in an unstructured text [16]. It falls under the category of unsupervised machine learning and probabilistic modeling [17]. *Latent Dirichlet allocation* (LDA) algorithm, a type of topic modeling, considers the input text as a bag of words and uses Bayesian inference to learn the distribution for topic grouping. Due to its ease of use, it has found application in a number of areas. Asuncion et al. applied the topic modeling technique to solve the problem of software artifact traceability [18]. Linstead et al. applied the topic model to extract 'concepts' or functional topics from the code[19]. In addition, a few of the other varied applications include potential for speech recognition[20], understanding social network ecosystems[21], investigating software evolution[22], analyzing genomic data[23], etc. The essence of the algorithm is its power to extract the semantic nature of the text. However, the challenge is then to analyze the meaning of the latent topics that emerge. Different automatic labeling algorithms have been proposed to solve this issue [24, 25]. Hindle et al. performed automatic topic labeling by assigning labels to the topic groups generated by the LDA algorithm by matching with a predefined word list. These word lists were derived from the ontology for non-functional requirements designed by prior research, and the ISO9126 list. This provided a methodology to perform domain specific labeling. There are some limitations to LDA. One of the limitations is the requirement that the number of topics must be predefined before running the algorithm. If the number of topics is small then the topic clusters formed are too generic, while if they are large then terms overlap between the topic clusters. Another limitation is that interpretation of the topic cluster semantics is left to the user.

2.2 Design Science Research Methodology

Design Science Research Methodology (DSRM) is common in the field of information system. However, it is rarely found in system engineering. Design science is a problem-solving research paradigm and results in the creation of an artifact to solve the problem. Hevner et al. proposed the design-science paradigm for information system (IS). IS being an applied science field, its research often contributes by creation of "new and innovative artifacts" [26]. The authors outlined the boundaries of DSRM by mentioning 2 processes and 4 artifacts. The processes are "design and evaluate." The artifacts are "constructs, models, methods, and instantiations" [26]. The primary purpose of artifacts created using DSRM is to "address unsolved problems" and the value is based on the "utility provided in solving those problems" [26].

2.3 National Vulnerability Database (NVD)

The National Vulnerability Database [27] produced by NIST Computer Security Divisions contains publicly available vulnerability data. Currently, there are over 44,000 Common Vulnerabilities and Exposures (CVE) entries in NVD. NVD has been used in prior research for developing vulnerability detection/discovery models or VDM[28]. In another related study, NVD data has been utilized by Schryen et al. to propose a software metrics named 'Patch index'[29]. They used NVD data to compare the security of the open source versus closed source software, by categorizing the vulnerability into three classes (high, low, and medium) and calculating the patch index based on number of patched versus un-patched vulnerabilities. NVD has also been used to build decision models that can be supplemented by user input, such as context information to allow for resource and effort management, through prioritization [30]. Fruhwirth et al. propose a security model where the temporal and environmental context information is provided by the practitioners, which allows for recalculation of the CVSS score and re-prioritization of the vulnerabilities. Chung et al. demonstrated a procedure for security risk assessment by cross-referencing the vector composed of the attributes "Data, Server and Application" with the vector "confidentiality, integrity, and availability"[31]. Another research used NVD data to propose a similarity based measurement that can be used for patching, mitigation and other security management areas [32]. In this research, An et al. utilized NVD along with Ontology for Vulnerability Management (OVM) to determine similarity between vulnerability using the hierarchical similarity algorithm. The characteristics considered for the similarity algorithm were "Type, Product, Access Vector, Access Complexity, Authentication, Confidentiality Impact, Integrate Impact and Availability Impact" [32].

NVD data contains information about different vulnerabilities in COTS, such as database servers, application servers,

browsers, etc. We integrate this information to determine the overall security composition of a custom software product and its potential implementation environment.

3. RELATED WORK

In a recent research, Neuhaus et al. used topic modeling to study the national vulnerabilities database (NVD) entries and identify emerging trends[33]. They found that classification done by the topic modeling technique LDA mostly agreed with the categorization done by MITRE. In this paper, we follow this approach to identify topic clusters from NVD entries. The rationale behind choosing an unsupervised learning problem such as LDA is that not all categories have been labeled as per CWE (Common Weakness Enumeration) category [15] in NVD. Using LDA, our goal is to discover groups of similar vulnerabilities within the data by clustering as found by Neuhaus et al. [33]. The benefit of using topic modeling for creating a risk assessment framework is it can be used to evaluate results from multiple different vulnerability scanners which produce textual output. Each scanner may have its own description of the vulnerabilities, which can be categorized into topics by using the topic modeling technique and assessed against the model created using the NVD entries.

In a prior research, Wang et al. proposed an ontology-based security assessment of the software products [7]. The products evaluated in the paper as a case study were web browsers such as Mozilla Firefox 3 and Internet Explorer 7. The two key questions in the paper were: "1) given a software product, what is the trustworthiness of it? 2) Among the similar products, which one is the best product in terms of security?" The questions that we propose in this research are similar but the software products are different. The software products we plan to evaluate are custom web-based, database-driven software applications. The security scores for these products are not likely to be present in NVD and need to be derived based on categories of vulnerability found. This research plans to fill this gap by proposing a framework for calculating the score of a custom database-driven software system.

Mkpong-Ruffin et al. (2007) proposed a software security risk assessment model using k-means clustering algorithm to group the vulnerability, and then through CVSS scores calculated the loss expectancy for each group [1]. The average CVSS scores were then calculated using the average growth rate for each month for the selected cluster. Finally, using the growth rate along with the average CVSS score, the predicted impact value was calculated for each cluster. They validated this proposed framework by generating a list of vulnerabilities using Microsoft's Threat Analysis and Modeling Tool (TAMT) and comparing with their own model. Their procedure proposed predictions of loss expectancy during design time using modeling tool output and the risk model built by them. The goal of our research is also to build such a quantitative security risk model; we plan to use the topic modeling technique instead of k-means clustering.

The procedure used in this research is similar to that used by Houmb et al.[3] The procedure was evaluated using a case study to demonstrate the security risk level assessment of a network. The procedure followed was: "Step 1: Identify vulnerabilities and potential fault introduction sources. Step 2: Estimate frequency and impact of vulnerabilities using CVSS. Step 3: Derive risk level from frequency and impact estimates" [8]. Our research uses a similar procedure but applies to a software product.

In a related research by Sui et al., the authors developed a framework for software security metrics score calculation based on quantitative criteria for integrality, availability and confidentiality [11]. They used the vulnerability scanner Nessus to generate a list of vulnerabilities and compare against their custom quantitative scoring criteria to calculate a security score. In our research, we also plan to use a vulnerability scanner, except we plan to use the impact score and threat likelihood value derived from NVD data to generate the security score.

Table 1 provides a brief overview of prior work on which we have based our paper.

Table 1. Prior Research in Quantitative Software Security Risk Assessment

	[1]	**[2]**	**[3]**	**[4]**	**[5]**
Name	SSRAM (Software Security Risk Assessment Model	Security Metrics for Software System	Risk Level using CVSS estimates of frequency and impact	SSAS (Software Security Assessment System)	VEXA : Software Vulnerability Extractor and Analyzer
Data Source	NVD	NVD	NVD	NVD	OSVDB and NVD
Technique	K means Clustering	Custom Calculation based on CVSS BASE SCORE and Number of Months(Frequency)	Bayesian Belief Network (BBN) topology	Custom Calculation based on CVSS and VRSS	Sum the impact and exploitability score. Apply weights based on the relevance of components
Application	In Design Phase	Security metrics score calculation for products such as: IE6 , Mozilla etc	Operation	Operation	Comparing Components(server, OS, database etc) for web serving systems
Evaluation	Case study	Experiment	Case study	Experiment	Case Study
Quantitative metrics	Predicted Loss Expectancy Predicted Impact Score Predicted	Base Impact, And Frequency	Frequency and impact score	Base Impact, Exploitability	Exploitability and impact level from CVSS

	[1]	[2]	[3]	[4]	[5]
	Frequency of Occurrence				
Tools used	TAMT, SQL Server Integration services, MS Analysis studio	None	Hugins, Nessus,	Nessus	None
Artifact Produced	Model	Equations	Method, Equations,	Methods, Equations, Tool	VEXA tool

4. Methodology Rationale

In this section we provide a rationale for the selection of Design Science Research Methodology (DSRM). The framework for our research has been shown in Figure 1.

4.1 Overview

Our research is carried out using guidelines provided by Hevner et al. [26] and presented using the steps proposed by Peffers et al. [37] for this methodology. Design Science (DS) research methodology is common in the field of information system. Hevner et al. categorized artifacts created through DS methodology into constructs, models, methods and instantiation. Table 2 provides examples of each of these artifact categories.

Table 2: Artifacts for Design Science Research [6]

Artifacts	Example
Constructs	vocabulary and symbols
Models	abstractions and representations
Methods	algorithms and practices
Instantiations	implemented and prototype systems

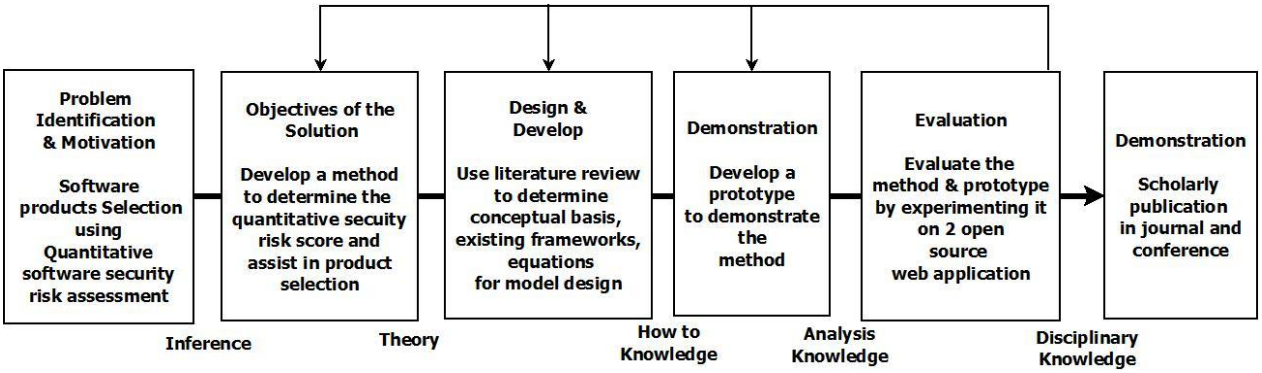

Fig 1. Design Science Research Process adapted from [7]

4.2 Rationale for DSRM

Design science research is very similar to action research since both attempt to identify and solve a problem. However, there are important differences between the two. Action research is a change-oriented approach that incorporates close collaboration between researcher and practitioners [38]. Design science research does not assume such collaboration. In design science research methodology (DSRM), the researcher identifies the issue, develops the theory and finally produces an artifact using the theory to solve the problem. Unlike DSRM, the initiator of the study in action research is often the practitioners. Our research is based on solving a problem by forming a procedure. The practitioners are not involved and therefore, DSRM is an appropriate methodology to follow.

As noted by Peffers et al., "Design Science is of importance in a discipline oriented to the creation of successful artifacts" [39]. The primary purpose of artifacts created using design science is to "address unsolved problems" and the value is based on the "utility provided in solving those problems" [26]. Since the goal of our research is to create a method to select between multiple software products based on quantitative software security risk assessment, DSRM is an ideal methodology to follow. Finally, design science is a problem-solving paradigm and thus supports our goal of solving the issue of selecting secure OSS products.

5. DSRM Steps

5.1 Problem Identification and Rationale

Recently several quantitative security risk assessment models have been proposed, many of which use vulnerability data from public databases such as NVD, OSVBD, etc. Even though vulnerability data has been used for predicting exploits, managing patching, and evaluating network security risks, it has rarely been applied to software selection. Since vulnerability is a type of defect that reduces the reliability of the system, filling this gap would help in the evaluation and selection of custom or open source software systems.

Existing quantitative software security risk models fall short since they evaluate individual COTS product and topology made of COTS, like servers, databases, etc. Some of them have clustered the vulnerability and applied it to the software design process[6]. However, in our review, we have not found any instrument providing objective measurement to help with

assessment of candidate software products such as web-based application.

The rationale for this research is to provide an artifact that can assist practitioners in making informed decisions based on empirical data and objective value. The primary research questions (RQ) evaluated were:

RQ1: What are the essential dimensions for evaluating a candidate software product before selection?

RQ2: How do we analyze the security risk level of a potential software product?

RQ3: Can we use a quantitative security risk assessment model to compare and select between custom software products?

5.2 Objective of the Solution

The objective of the solution is to provide a method using the quantitative security risk assessment model based on prior research and to demonstrate its use as an instrument that can provide quantitative value for objective decision making in software selection.

5.3 Design and Development

The proposed method consists of the following steps:

1) Collect NVD data and cluster using topic modeling methods
2) Obtain the categories of vulnerability that the vulnerability scanner tool can report
3) Label the topics found in step 1, using the categories from the vulnerability scanner. This results in the quantitative security risk assessment model
4) Inspect and find system requirements for the candidate software product. Identify the operating system, database servers, web servers and other components needed, their brand and version
5) Determine the security score of the system requirement components
6) Next for each product (assuming OSS product), download the code and perform static analysis. Use the quantitative security risk assessment model to determine impact score and frequency
7) Next install the OSS and perform dynamic analysis using web application scanner. Use the quantitative security risk assessment model to determine impact score and frequency
8) Repeat this for all candidate products and compare the results.

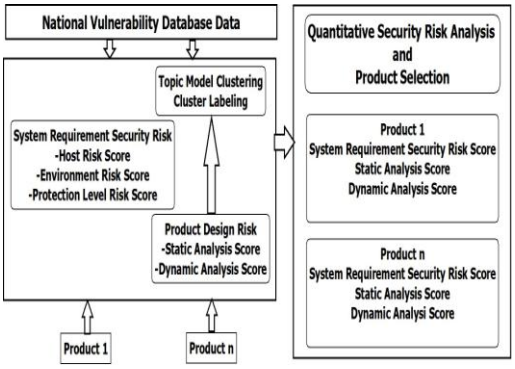

Fig 2. Model for OSS selection using Quantitative Security Risk Analysis

Figure 2 shows the design of the model for OSS selection using quantitative security risk analysis. To design the artifacts, we first investigated the sources of security risk of a software product. A software product has security risk due to the code base, the COTS product (servers, OS, etc.) it is hosted on and the environment it is installed on. Thus, we propose two primary dimensions for evaluating the security risk of the candidate software product: *System Requirement Risk* and *Product Design Risk,* as shown in Figure 3.

5.3.1 Dimensions of Risk

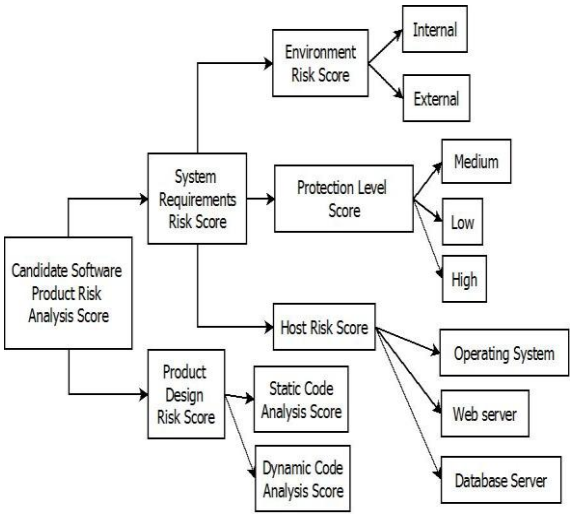

Fig 3. Risk assessment dimensions for the candidate product

5.3.1.1 System Requirement Risk

A software product comes with system requirements, which specify the operating system, database server and web server on which it can operate. These are sources of risk and can be further subdivided into:

Host Risk: This is the first sub-dimension and is related to the COTS product on which it is hosted. These are operating systems (OS), web servers and database servers. Each of them is a source of security risk based on the number of vulnerabilities that reside within.

Environment Risk: The next risk sub-dimension is due to the environment in which the software product might be installed. A public facing server is more risky than one installed within the firewall and closed to public access.

Protection Level Risk: The final risk sub-dimension is a subjective score that the practitioners can assign based on knowledge about the protection controls that exists in terms of people, software and process on the host.

5.3.1.2 Product Design Risk

The second dimension of risk is due to the flaws in the software caused by programming errors such as integer overflow or buffer overflow. These can be detected using static and dynamic analysis to identify a list of potential vulnerabilities.

The next section describes the quantitative security risk assessment model development process.

5.3.2 Development

Next, we obtain the empirical data to develop the quantitative security risk model. The development process consists of 2 primary tasks: data collection and model building. To build the model, we grouped the vulnerabilities into clusters using Topic Modeling and then estimated the impact using the average base score, and severity using the number of vulnerabilities in each cluster.

5.3.2.1 Data Collection

The empirical data for building the risk model was obtained from the National Vulnerability Database (NVD). The data is available as a XML file named nvdcve-2.0-[year].xml, where the year indicates a number from 2002 to 2012. We built a perl script to extract the cve_id, published date, score and summary field and loaded it to a database server. In addition, a CSV file was created with one line per CVE entry. We refer to this file as the extracted NVD file in the rest of this paper.

5.3.2.2 Topic Modeling

We applied the Latent Dirichlet allocation (LDA) technique to the extracted NVD file using Stanford Topic Modeling Toolbox[37]. Known as a type of unsupervised machine learning technique, LDA considers the input text as a bag of words and uses Bayesian inference to learn the distribution for topic grouping. To build the LDA model, we applied a simple English tokenizer and removed English stop words like 'the,' 'of,' etc. We considered a summary text with 2 or more terms. We filtered out terms that appear in less than 4 topics.

One of the parameters essential for LDA is "number of topics." We ignored non-words and non-numbers. We ran the LDA model using the CVB0 algorithm, looping over the number of topics ranging from 10 to 100 and calculating the perplexity score. The model was trained using 80% of the data and tested using 20% of the data. Finally, we chose 40 as the number of topics since it gave the least amount of overlap and least amount of generic topics. Figure 4 shows the graph demonstrating the choice of optimum topics number.

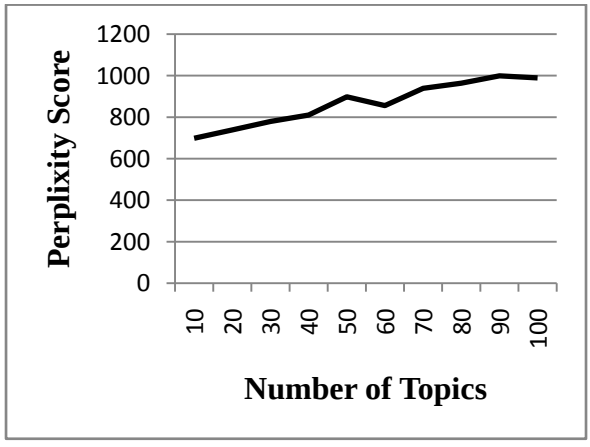

Fig 4. Topic Selection Evaluation

The output of this process resulted in multiple files of which we used only two. The first file contained the topic groups and associated terms as shown in Table 3. The second file contained indicators showing association between the CVE entry and the associated topic distribution.

Table 3. Topics and Top 10 Related Terms Output From LDA toolbox

Topic	Terms
Topic 24	string, function, arbitrary, execute, allows, attackers, format, remote, commands, shell, vulnerability, code, metacharacters, properly, handled, command, variable, argument, program, functions, earlier, demonstrated, specifiers, using, cgi, context-dependent, variables, query, php, certain, strings, arguments, allow, line, option, expression, used, perl, regular, possibly
Topic 26	server, web, remote, attackers, allows, http, request, url, apache, source, redirect, determine, iis, manager, requests, sites, page, application, open, uri, default, proxy, network, node, tomcat, jsp, mail, interfacepages, administration, sap, scripts, coldfusion, existence, error, contains
Topic 23	script, scripting, web, xss, cross-site, html, inject, arbitrary, attackers, remote, vulnerability, allows, parameter, earlier, index.php, search, page, action, parameters, cms, query, search.php, uri, error, title, _info, msg, path, portal, different, insert, login, theme, keyword

The extracted NVD file and the output from the LDA toolbox are then loaded into the database server for calculation of estimated score and the frequency of the topic group. We used SQL Server Express for data storage and manipulation. The extracted NVD file is loaded into a table with attributes cve_id, published date, score and summary. The output from the LDA toolbox is also loaded into the database server. This data is parsed so that we have a table showing the relationship between CVE_ID, Topic Name and Associated Probability Distributions. Since there are many topics per CVE, we need to determine the most relevant topic. For each CVE, we select the topic with the highest probability distribution. This gives us a vector consisting of CVE_ID, Topic Name, Score and maximum probability distribution as shown in Table 5.

Table 5: Vulnerability and Topic with maximum probability distribution

cve_id	Topic Name	Score	Max Probability
CVE-2008-2012	Topic 02	7.5	0.8308502
CVE-2012-0005	Topic 28	6.9	0.8247771
CVE-2012-0011	Topic 08	9.3	0.8780806
CVE-2012-0018	Topic 08	9.3	0.9353617
CVE-2012-0019	Topic 08	9.3	0.9687625
CVE-2012-0020	Topic 08	9.3	0.9687625
CVE-2012-0056	Topic 19	6.9	0.8364574
CVE-2012-0069	Topic 02	7.5	0.9886265
CVE-2012-0072	Topic 27	5	0.9918746

5.3.2.3 Labeling the topics

The LDA toolbox generated the topics and the associated terms but their interpretation remained to be done. A number of algorithms has been presented in order to interpret the topic clusters. Hindle et. al [25] generated the word list with category names, using different sources and then labeled the topic if any of the words from the word list categories matched the topic terms.

We attempted to assign labels using 2 methods. In the first method, for each topic cluster, we counted the number of CVE per CWE. We labeled a topic cluster with a CWE name if it had a maximum number of CVEs of that type. In the second method, we obtained the categories from our vulnerability scanner and applied the same topic model created earlier using NVD topic to assign them to topic groups. For each vulnerability scanner category, we found the topic that represented the highest probability distribution. Table 4 is a sample of the results for this method, for topics shown in Table 3. Both attempts were not very accurate and needed manual intervention. In both cases, manual review was needed to fix a certain percentage of the topics.

Table 4. Mapping of Topic to Skipfish Vulnerability

Topic	Skipfish Reported Vulnerability
Topic 24	Format string vulnerabilities.
Topic 26	Explicit SQL-like syntax in GET or POST parameters.
Topic 23	Stored and reflected XSS vectors in document body (minimal JS XSS support present).

5.4 Demonstration and Evaluation

To evaluate the measurement tool, we selected open source software *openEMR*[12] and *openMRS*[13] as candidate software products. We selected open source products belonging to same function domain. Both of them are open source health record systems. *openEMR* is developed in PHP with MySQL as the backend. *openMRS* is developed in java with MySQL as backend. Both run on a MySQL database server. We setup the systems in our test environment: one on Apache Http server and the other on Apache Tomcat server, as specified by their respective system requirement documentation. The candidate systems were scanned for vulnerabilities using the open source web application scanner *skipfish[41]*. The host operating system for both the software products was Windows 7.

5.4.1 System Requirements Risk Analysis

The steps in the process are as follows:

Step 1: Inspect system requirements and identify major COTS components such as database server, web server, etc.

1. For openMRS, we identify the following system requirements to host the product: Apache Tomact 6 and MySQL 6

2. Find base score; exploit score and the number of vulnerabilities for each component

.

An example for the component MySQL is given in Table 6.

Table 6: Sample component base score, exploit score and the number vulnerabilities

CVE ID	Base score	Exploit score
CVE-2004-0388	2.1	3.9
CVE-2004-0627	10	10
CVE-2004-0628	10	10
CVE-2005-1636	4.6	3.9
CVE-2005-2558	4.6	3.9
CVE-2005-2572	8.5	6.8
CVE-2005-2573	5	10
CVE-2006-0369	2.1	3.9
CVE-2006-0903	4.6	3.9

1. Calculate the component security score as follows using the impact score formula proposed by Sui et al.[4]

$$\text{Component Security Risk Score} = \frac{\sum_{i=1}^{n} Base\ Score\ X\ Exploit\ Score}{n} \quad (1)$$

Where n = number of vulnerabilities in the component

In addition we need to consider the exposure score. An OSS product installed on a public facing server is more vulnerable than on a private server. Therefore, we assign a weight to each based on exposure.

Finally, we need to consider the protection level of the environment where it will be installed. We provided three values for practitioners to select from based on their professional judgment.

Total System Requirement Security Risk score = Component Score * Exposure Weight * Protection Level

Table 7. openMRS System Requirements Risk Score

Host Components	Security Risk Score	Exposure Weight	Protection level	
Apache Tomcat 6	C1 =41.63	External E1=2	Low P1=3	C1*E1*P1 = 249.8
MySQL 6	C2=33.81	Internal E2=1	High P2=1	C2*E2*P2 = 33.81
Apache Windows 2008	C3=13.33	External E3=2	Medium P3=2	C3*E3*P3 = 53.32
Tomcat Windows 2008	C4=13.33	Internal E4=1	High P4=1	C3*E3*P3 = 13.33
Total System Requirement Security Risk score				350.26

Table 8. openEMR System Requirements Risk Score

Host	Component Score	Exposure Weight	Protection level	
Apache 2.2	37.59	External=2	Low=3	225.54
MySQL 6	33.81	Internal=1	High=1	33.81
Apache Windows 7	13.33	External=2	Medium=2	53.32
Tomcat Windows 7	13.33	Internal=1	High=1	13.33
Total Component Security Risk score				326.00

Practitioners can also perform an if-then analysis by changing the operating system type, exposure score and protection level.

5.4.2 Product Design Risk Analysis

To find the security risk due to coding flaws, we conducted static and dynamic analysis.

5.4.2.1 Dynamic Analysis

We conducted dynamic analysis using a web application scanner for the candidate products. Below is a list of representative vulnerabilities found by dynamic analysis of openMRS using SkipFish[41].

Table 9. openMRS Dynamic Analysis

Name of the Representative Threat	**Number of Vulnerabilities Found in Candidate 1 (openMRS)**
Integer overflow vector	4
Incorrect or missing charset	7
Incorrect or missing MIME type	1
HTML form with no apparent XSRF protection	2

We found that some of these had direct mapping to CWE but others did not, for example, integer overflow maps to *CWE*-190 whose parent is CWE-189 - Numeric Errors. Certain vulnerabilities in NVD have been mapped to CWE-189 - Numeric Errors, thus enabling us in extracting average impact score and frequency score value. Others such as 'Incorrect or missing charset' had no mapping in CWE used by NVD. However, using the previously trained topic model, we were able to assign a topic to each of the skipfish vulnerability types.

Table 10. Assignment of a topic to each of the skipfish vulnerability types

Name of the Representative Threat	**Topic**
Integer overflow vector	Topic 37
Incorrect or missing charset	Topic 01
Incorrect or missing MIME type	Topic 00
HTML form with no apparent XSRF protection	Topic 01

Next we applied the method outlined by Wang et al. [35] to find the security metric score.

Count the number of vulnerabilities per topic and the frequency as shown in Tables 11 and 12.

Table 11. Topic vulnerability count and date range

Topic	Vulnerability Count	Min Date	Max date
Topic 01	1153	2/14/1998	5/24/2012
Topic 30	720	11/11/1988	5/24/2012
Topic 02	5083	11/1/1995	5/23/2012

Table 12. Topic vulnerability count and Frequency

Topic Name	Vulnerability Count V	Number of Months Min Date-Max Date M	Frequency Per Month V/M
Topic 00	604	187	3
Topic 01	1153	171	6
Topic 37	1393	152	9

Average Impact Score for each Topic Cluster is calculated as:

$$\text{Average Score} = \sum_{i=1}^{n} base\ Score_i$$

Where n=number of vulnerability in the Topic.

Table 13. openMRS Topic Average score and Frequency

Topic Name	Average Score	Frequency Per Month
Topic 00	5.812582	F1=3
Topic 01	5.763399	F2=6
Topic 37	7.635104	F3=9

Weight in Table 14 is the Number of Vulnerability Found in Candidate 1 for each topic.

Table 14: openMRS vulnerability weight analysis

Topic Name	Average Impact Score S	Probability P	Weight (W)
Topic 00	S1=5.81	P1=F1/F1+F2+F3=0.17	W1=1
Topic 01	S2=5.76	P2=F2/F1+F2+F3=0.33	W2=9
Topic 37	S3=7.64	P3=F3/F1+F2+F3=0.5	W3=4

Product Design Risk Score= S1*P1*W1 + S2*P2*W2 + S2*P2*W2 = 33.53

5.4.2.2 openEMR

Next, we evaluate the second candidate software product. This product has a different system requirement risk score since it is installed on a different web server. The Table 15 shows a sample out obtained through dynamic scanning of openEMR.

Table 15: Sample openEMR Dynamic Analysis

Name of the Representative Threat	**Number of Vulnerabilities Found in Candidate 1 (openEMR)**
Integer overflow vector	4
Incorrect or missing charset	8
Incorrect or missing MIME type	1
HTML form with no apparent XSRF protection	2

In the case of openEMR, for product design risk analysis, the types of vulnerability categories reported in our experiment remained the same. Thus, only the number of vulnerabilities (weight w) reported was different.

Table 16. openEMR vulnerability weight analysis

Topic Name	Average Score S	Probability P	Weight (W)
Topic 00	S1=5.812582	P1=0.166667	W1=1
Topic 01	S2=5.763399	P2=0.333333	W2=10
Topic 37	S3=7.635104	P3=0.5	W3=4

Security Metric Score= S1*P1*W1 + S2*P2*W2 + S2*P2*W2 = 35.45

Finally at the end of the procedure, we have the following security scores for the two candidate products for comparison.

Table 17.Quantitative Security Risk evaluation of openMRS

System Requirement Risk	350.26
Product Design Risk	
Static Analysis	100
Dynamic Analysis Risk	33.53

Table 18.Quantitative Security Risk evaluation openEMR

System Requirement Risk	326
Product Design Risk	
Static Analysis	100
Dynamic Analysis Risk	35.45

The scores of the individual dimensions were not aggregated since they might end up being the same on aggregation. Individual comparison of each dimension is suggested.

6. Results and Discussion

In order to evaluate the artifact, we needed to detect the potential vulnerabilities in the candidate software product. However, this was challenging for multiple reasons. Much effort and skill is required for manual vulnerability detection. Automated vulnerability testing is faster but suffers from problems related to false positives and missed issues. The two methods of automatic vulnerability detection are static analysis and dynamic analysis. Static analysis is performed by examining the source code or the binaries of the software. Dynamic analysis is performed on the application when it is running, using a vulnerability scanner. The challenge with vulnerability testing is that neither of the methods is comprehensive by itself, as pointed out by Austin et al. [13]. In addition, the output of different scanners produces different results and often misses some vulnerability.

We selected the method of automated dynamic vulnerability detection for our experiment. As per Austin et al., in case of limited time, "automated penetration testing can be conducted to discover penetration bugs." Our assumption is that we will use the same dynamic analysis tools on both candidate products, causing similar results in both. However, we recognize that usage of the tool and code-based characteristics of candidate software products have an impact on the results of the static analysis tool[42]. The goal of the research is to evaluate relative security risk level of the products and not the absolute security risk level. Since we have not found any absolute solution so far for vulnerability detection, we proceed with the automated vulnerability detection method. However, we acknowledge that detection of vulnerability is an issue as identified by [13, 43] and others.

We examined the topic groups and found that they presented the vulnerability domain very well, thereby confirming the results presented by Neuhaus et al. [33]. We encountered multiple challenges when implementing the assessment model for software selection procedure. First, we wanted to design the proposed procedure such that the practitioners using the procedure must find it practical and not overly time consuming. In an empirical study of how OSS is selected in small companies, authors [44] find that OSS selection procedure is sequential , with the 'first fit' solution being selected rather than 'best-fit'[44]. Thus we assume that practitioners review multiple tools in a short duration of time, leaving very little time for extensive security testing. Therefore, we selected automatic penetration testing methods for vulnerability detection since it is time-efficient for detecting implementation flaws[13]. Second, finding a method to effectively create a list(s) of vulnerabilities for a candidate software system has been difficult due to issues reported in prior research [13, 42, 45]. Manual testing depends on the skills and knowledge of the tester, including the fact that it is time consuming. Automated vulnerability detection has issues due to limitations of the vulnerability analysis tools[42], a high percentage of false positives, lack of coverage due to inability to scan all pages, etc. We argue that some evaluation is better than none. At least the software systems would not be selected without any kind of security risk evaluation. Finally, we were faced with the issue of matching the results of different scanners with the quantitative security risk assessment model. The results of static analysis as well as dynamic analysis were in different textual formats. The documentation of different vulnerability scanners contains a long list of individual security problems detected by the scanners. It might have been better to manually map these problem categories to topics, rather than doing this with text mining and matching the topics. In addition, since one scanner and one methodology will not find all vulnerabilities, we

propose the use of multiple tools for scanning and thus had multiple lists. This added the complexity of converting the text output of the security scanners to a format that could be used to infer the topic and then obtaining the impact score and frequency for that topic.

We began by investigating 3 questions:

RQ1: What are the essential dimensions for evaluating a candidate software product before selection?

The security risk level of a potential software product can be evaluated by considering two dimensions: System Requirement Dimension and Product Design dimensions. Each of these has sub dimensions and steps as described in Section 5.3.1. Following 'System Thinking,' we evaluate the product, its environment and its host.

RQ2: Can we use a quantitative security risk assessment model to compare and select between custom software products?

We can build a quantitative security risk assessment model using empirical data about vulnerabilities from NVD, as demonstrated in prior research and in Section 5.3.2 of this paper. We used topic modeling to determine groups of threats, their likelihood and impact score.

RQ3: How do we analyze the security risk level of a potential software product?

We analyze the security risk level of a potential software product using our defined dimensions and steps similar to 800:53 IT risk assessment [46]. This has been described in Section 5.4 through an experiment.

7. Limitations

The topic clusters generated by machine might have overlap between them. This has not been handled in this research. The assessment model will only report on known vulnerabilities since it is based on historical data. Most users might be more worried about unknown vulnerabilities than known security risks. However, known vulnerabilities still represent a significant percentage of threats to a software system. Scholte et al. conducted an empirical study of vulnerabilities for web-based software systems and found that complexities of these vulnerabilities are not increasing over time. Most of these vulnerabilities are happening due to simple reasons, such as lack of input validation [47]. In a study comparing the effectiveness of different vulnerability scanners, authors Bau et al. found that different scanners are strong in different categories [48]. Also, they found that vulnerabilities detected by scanners in popular categories such as SQL injection and cross-site scripting is proportional to those found in the real situation. They lack in finding second order vulnerabilities such as "Cross-Channel Scripting, Cross-Site Request Forgery, and Malware Presence" [48]. An area of future work is to evaluate a better method for automatic labeling of the topics.

8. CONCLUSION

The aim of this research is to propose an artifact to support the evaluation of off-the-shelf software products before adoption using a quantitative security risk assessment model. We propose the application of the topic model to discover latent groups of vulnerabilities and develop a risk model based on those groups. This quantitative security risk model combined with component, static, and dynamic analysis output serves to evaluate multiple software products. We acknowledge the limitation of vulnerability scanning tools and its impact on the procedure. However, given the time constraints that practitioners have to evaluate a software system and the lack of better methods for objective analysis, we propose that our model is a support for a quick and preliminary analysis which can be further supplemented by more intensive analysis using security experts, depending on the context and available time. For future work, we propose investigating other topic modeling algorithms and the possibility of applying automatic labeling to the topic groups. We believe that the addition of such features to the assessment model would assist in the development of a user-friendly tool to assist the practitioners in comparing multiple products. Topics have not been manually or automatically labeled. Having topic clusters labeled would provide the practitioners an enhanced accuracy and more practical way to judge the score(s) and determine their mitigation strategies.

9. Acknowledgement

The material is summarized from parts of the dissertation to be submitted to The George Washington University in partial fulfillment of the requirements for the Doctor of Philosophy degree.

10. REFERENCES

[1] J. L. Bayuk, "Systems Security Engineering," Security & Privacy, IEEE, vol. 9, pp. 72-74, 2011.

[2] INCOSE, "'INCOSE Systems Engineering Handbook," vol. 3.2.

[3] D. Childs, "Information technology security system engineering methodology," in Aerospace Conference, 2003. Proceedings. 2003 IEEE, 2003, pp. 3393-3401.

[4] N. F. Schneidewind, "Methods for assessing COTS reliability, maintainability, and availability," in Software Maintenance, 1998. Proceedings. International Conference on, 1998, pp. 224-225.

[5] R. Khan, "Secure software development: a prescriptive framework," Computer Fraud & Security, vol. 2011, pp. 12-20, 2011.

[6] I. Mkpong-Ruffin, D. Umphress, J. Hamilton, and J. Gilbert, "Quantitative software security risk assessment model," presented at the Proceedings of the 2007 ACM workshop on Quality of protection, Alexandria, Virginia, USA, 2007.

[7] J. A. Wang, M. Guo, H. Wang, M. Xia, and L. Zhou, "Ontology-based security assessment for software products," presented at the Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, Oak Ridge, Tennessee, 2009.

[8] S. H. Houmb, V. N. L. Franqueira, and E. A. Engum, "Quantifying security risk level from CVSS estimates of frequency and impact," Journal of Systems and Software, vol. 83, pp. 1622-1634, 2010.

[9] W. Zhihu and W. Xin, "Research on technologies in quantitative risk assessment and forcast of network security," in Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, 2010, pp. V6-524-V6-528.

[10] H. Joh, "Quantitative analyses of software vulnerabilities," Ph.D. 3489881, Colorado State University, United States -- Colorado, 2011.

[11] Y. L. Chenmeng Sui, Yun Liu, "A Software Security Assessment System Based On Analysis of Vulnerabilities," Journal of Convergence Information Technology, vol. 7, p. 211 ~ 219, 2012.

[12] S.-W. Woo, H. Joh, O. H. Alhazmi, and Y. K. Malaiya, "Modeling vulnerability discovery process in Apache and IIS HTTP servers," Computers & Security, vol. 30, pp. 50-62, 2011.

[13] A. Austin and L. Williams, "One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques," in Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, 2011, pp. 97-106.

[14] W. J. Sung, J. H. Kim, and S. Y. Rhew, "A Quality Model for Open Source Software Selection," in Advanced Language Processing and Web Information Technology, 2007. ALPIT 2007. Sixth International Conference on, 2007, pp. 515-519.

[15] MITRE. (6/24/2012). CWE -Common Weakness Enumeration. Available: http://cwe.mitre.org/compatible/category.html

[16] "Probabilistic Topic Models," Communications of the ACM, vol. 55, pp. 77-84, 2012.

[17] G. Anthes, "Topic Models Vs. Unstructured Data," Communications of the ACM, vol. 53, pp. 16-18, 2010.

[18] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in Software Engineering, 2010 ACM/IEEE 32nd International Conference on, 2010, pp. 95-104.

[19] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining concepts from code with probabilistic topic models," presented at the Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA, 2007.

[20] C. Kuan-Yu, C. Hsuan-Sheng, and B. Chen, "Latent topic modeling of word vicinity information for speech recognition," in Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on, 2010, pp. 5394-5397.

[21] R. A. Negoescu and D. Gatica-Perez, "Modeling Flickr Communities Through Probabilistic Topic-Based Analysis," Multimedia, IEEE Transactions on, vol. 12, pp. 399-416, 2010.

[22] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Validating the Use of Topic Models for Software Evolution," in Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on, 2010, pp. 55-64.

[23] C. Xin, H. Xiaohua, S. Xiajiong, and G. Rosen, "Probabilistic topic modeling for genomic data interpretation," in Bioinformatics and Biomedicine (BIBM), 2010 IEEE International Conference on, 2010, pp. 149-152.

[24] D. Magatti, S. Calegari, D. Ciucci, and F. Stella, "Automatic Labeling of Topics," in Intelligent Systems Design and Applications, 2009. ISDA '09. Ninth International Conference on, 2009, pp. 1227-1232.

[25] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos, "Automated topic naming to support cross-project analysis of software maintenance activities," presented at the Proceedings of the 8th Working Conference on Mining Software Repositories, Waikiki, Honolulu, HI, USA, 2011.

[26] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," MIS Q., vol. 28, pp. 75-105, 2004.

[27] NVD. National vulnerability database. Available: http://nvd.nist.gov

[28] A. Ozment, "Improving vulnerability discovery models," presented at the Proceedings of the 2007 ACM workshop on Quality of protection, Alexandria, Virginia, USA, 2007.

[29] G. Schryen and R. Kadura, "Open source vs. closed source software: towards measuring security," presented at the Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii, 2009.

[30] C. Fruhwirth and T. Mannisto, "Improving CVSS-based vulnerability prioritization and response with context information," presented at the Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009.

[31] Y. J. Chung, I. Kim, N. Lee, T. Lee, and H. P. In, "Security risk vector for quantitative asset assessment," presented at the Proceedings of the 2005 international conference on Computational Science and Its Applications - Volume Part II, Singapore, 2005.

[32] W. Ju An, Z. Linfeng, G. Minzhe, W. Hao, and J. Camargo, "Measuring Similarity for Security Vulnerabilities," in System Sciences (HICSS), 2010 43rd Hawaii International Conference on, 2010, pp. 1-10.

[33] S. Neuhaus and T. Zimmermann, "Security Trend Analysis with CVE Topic Models," in Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, 2010, pp. 111-120.

[34] (06-01-2012). CVE -Common Vulnerabilities and Exposures (CVE). Available: http://cve.mitre.org/

[35] J. A. Wang, H. Wang, M. Guo, and M. Xia, "Security metrics for software systems," presented at the Proceedings of the 47th Annual Southeast Regional Conference, Clemson, South Carolina, 2009.

[36] N. Mendes, J. Duraes, and H. Madeira, "Benchmarking the Security of Web Serving Systems Based on Known Vulnerabilities," in Dependable Computing (LADC), 2011 5th Latin-American Symposium on, 2011, pp. 55-64.

[37] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," Journal of Management Information Systems, vol. 24, pp. 45-77, 2008.

[38] J. V. Juhani Iivari "Action Research and Design Science Research – Seemingly similar but decisively dissimilar," 17th European Conference on Information Systems, 2009.

[39] K. E. N. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," Journal of Management Information Systems, vol. 24, pp. 45-77, Winter2007/2008 2007.

[40] Stanford Topic Modeling Toolbox. Available: http://nlp.stanford.edu/software/tmt/tmt-0.4/

[41] (05/05/2012). Google Code. Available: http://code.google.com/p/skipfish/

[42] (05/05/2012). Software [In]security: Comparing Apples, Oranges, and Aardvarks (or, All Static Analysis Tools Are Not Created Equal) | As the Market Grows | InformIT. Available: http://www.informit.com/articles/article.aspx?p=1680863

[43] A. Doupe, M. Cova, and G. Vigna, "Why Johnny can't pentest: an analysis of black-box web vulnerability scanners," presented at the Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment, Bonn, Germany, 2010.

[44] O. Hauge, T. Osterlie, C.-F. Sorensen, and M. Gerea, "An empirical study on selection of Open Source Software - Preliminary results," presented at the Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009.

[45] L. Suto, "Title," unpublished|.

[46] A. G. Gary Stoneburner, and Alexis Feringa. (2002, 5/5/2012). Risk Management Guide for Information Technology Systems. NIST-SP 800:30.

[47] T. Scholte, D. Balzarotti, and E. Kirda, "Have things changed now? An empirical study on input validation vulnerabilities in web applications," Computers & Security, vol. 31, pp. 344-356, 2012.

[48] B. Jason, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," 2010, pp. 332-345.