Security and vulnerability analysis of web applications

Le, Ha Thanh

2011

Le, H. T. (2011). Security and vulnerability analysis of web applications. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/43438>

<https://doi.org/10.32657/10356/43438>

NANYANG
TECHNOLOGICAL
UNIVERSITY

SECURITY AND VULNERABILITY
ANALYSIS OF
WEB APPLICATIONS

LE HA THANH

SCHOOL OF COMPUTER ENGINEERING

2011

SECURITY AND VULNERABILITY ANALYSIS OF WEB APPLICATIONS

LE HA THANH

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

2011

ACKNOWLEDGEMENT

I sincerely thank Dr. Peter Kok Keong Loh for his support and guidance that have made this project possible. I would also like to thank Mr. Deepak Subramanian who worked along side me and provided active support to me. I am also thankful to the assistance provided by the final year students Mr. Hong Yang Tok, Mr. Somsak Jangdaboud and Mr. Seah Derek for the implementation of data conversion utilities for the project. I am thankful to Assoc. Prof. Hsu Wen Jing for allowing me to continue my work under him. I want to thank Mrs. Christina from Forensic and Security Laboratory for her technical help during I work in that lab. I finally thank the Nanyang Technological University for providing me the facilities and the opportunity to be part of this research program.

TABLE OF CONTENTS

Acknowledgement	2
List of Figures	5
List of Tables	6
Abstract	7
Chapter 1 Introduction	8
1.1. Background	8
1.2. Motivation	10
1.3. Objectives	11
1.4. Thesis organization	12
Chapter 2 Literature Study	13
2.1. Vulnerability Analysis	16
2.2. Vulnerability Description	29
2.3. Vulnerability Database and Taxonomy	31
2.4. Vulnerability Impact Evaluation and Scoring	33
Chapter 3 Web Application Vulnerability Analysis Framework	36
Chapter 4 Property-Based Vulnerability Description Model for Web Applications	38
4.1. Application Properties in Vulnerability Analysis	38
4.2. Describing Vulnerability Using Vulnerable Property Relation Graph (VPRG)	45
4.2.1. Property-Property Dependencies	46
4.2.2. Vulnerable Property Relation Graph (VPRG)	48
4.2.3. Security-related Operation State of Property	51
4.2.4. Matrix Form of VPRG – Vulnerable Property Relation Matrix (VPRM)	52
4.3. Impact Evaluation Using VPRG	54
4.3.1. Combination Relations of Cause-Properties	54
4.3.2. Distribution of Consequence and Consequence-Property Cardinality	59
4.3.3. Impact Estimation of Vulnerability Exploitation	60
4.3.3.1. <i>Impact Estimation</i>	60
4.3.3.2. <i>Impact Calculation</i>	62
Chapter 5 Experiments	65

5.1. Vulnerability Property and Property Relationship Database Preparation	66
5.2. Experiments.....	66
Chapter 6 Conclusion	72
6.1. Summary	72
6.2. Future Work	73
Bibliography and References	75

LIST OF FIGURES

Figure 1.	Flaw syntax represents installation integrity flaw ([24])	17
Figure 2.	Sources of vulnerability on Multi-tier Web Application architecture with external links and user-active customization.	21
Figure 3.	The framework for Web application vulnerability analysis.....	36
Figure 4.	Behavior Correlation Graph.....	42
Figure 5.	VPRG of vulnerability at p_v	49
Figure 6.	Conjunction (a) and disjunction (b) of properties in VPRG graph.	55
Figure 7.	Cause graph of vulnerable property p_v	56
Figure 8.	VPRM and VPRG of SQL Injection vulnerability: (a) CWE-89 and (b) described in SearchSecurity.com.	67
Figure 9.	VPRM and VPRG of Cross Site Scripting (CWE-79): (a) simple version and (b) Extended version with variants on condition properties.....	68
Figure 10.	VPRM and VPRG of Cross Site Request Forgery (CWE-352).....	68
Figure 11.	VPRM and VPRG Apache Chunked-Encoding Memory Corruption Vulnerability (CVE-2002-0392)	69

LIST OF TABLES

TABLE 1. VULNERABILITY ANALYSIS METHODOLOGIES.....	13
TABLE 2. VULNERABILITY DESCRIPTION.....	15
TABLE 3. VULNERABILITY IMPACT EVALUATION AND SCORING.....	16
TABLE 4. VULNERABILITY DETECTION MODELS AND ANALYSIS TECHNIQUES. ..	17
TABLE 5. VPRM REPRESENTS VPRG	53
TABLE 6. SEVERITY SCORE OF SQL INJECTION, XSS AND CSRF VULNERABILITY (CWE).....	70
TABLE 7. SECURE STATE OF CVE-2002-0392 VULNERABILITY PROPERTIES.....	71

ABSTRACT

Vulnerabilities in Web applications have grown exponentially over the last decade. For effective remediation, more comprehensive analytical models are necessary. This work introduces a property-based description model for Web-based vulnerability analysis. We present the theoretical basis of this property-based analysis model. The analysis of cause- and consequence- views of the model can support inference of the cause of vulnerability as well as the evaluation of vulnerability impact and severity. The effectiveness of the model is then demonstrated by its applications on real world vulnerabilities.

CHAPTER 1

INTRODUCTION

Many software applications contain certain vulnerabilities and can have their integrity compromised for untoward purposes. Security breaches have affected directly on system platforms or via a variety of related applications that run off these platforms. Web-based applications, in particular, are typically multi-tiered, comprising cross-platform components. Many of them have a clientele base that is large and wide spread with possibly many unknown users. From the application vendor's and service provider's view points, the Web provides a seamless and unlimited environment to deploy their applications which can easily reach their end users world-wide. The rapid increase in the number of corporate Web applications offers end users choices of several different applications. Moreover, the intrinsic characteristics of a Web-based application increase the variety of possible operations and interactions. Unavoidably, security occurrences have increased along with the exponential growth of the Internet. This raises many security issues and exacerbates the demand for practical, customer-friendly solutions [1-4].

1.1. Background

Normally, the process of securing applications involves analyzing the application architecture for logical mistakes and scanning the source code to detect errors that may lead to unsecured states. Besides, experts usually collect information about application exploitations people have experienced and create a database or establish taxonomy in order to classify the vulnerabilities and

related issues. Generally, vulnerability prevention can follow either a *forward* or *backward* approach. In the forward approach, a database of known vulnerabilities is used. The scanning process based on penetration testing techniques (e.g. active request-response analysis) will detect a vulnerability based on its known features. The scanned result may provide certain information about the possible exploitation consequences and the associated remediation. In addition, evaluation methods such as nCircle Vulnerability Scoring System [5], Adobe Severity Rating System for Security [6], Cenxio ARC-HARM score [7], analysis Daily Vulnerability Exposure (DVE) trend over time [8], Common Vulnerability Scoring System (CVSS) [9], Microsoft Severity Rating System [10], Secunia “Criticality” [11], and US-CERT vulnerability severity metric [12] may be used to estimate the severity and impact of vulnerability. This approach can be seen in several vulnerability scanners [13-15]. The scanners and vulnerability databases (such as NIST [16], CVE [17], Finjan [18], Secunia [19], SecurityFocus [20, 21], and US-CERT [22]) help users to detect the known exploited vulnerabilities of infrastructure systems such as out-of-date secure patch operating system or insecure version of Web server.

On the other hand, the backward approach starts with an exploitation report and tries to determine which vulnerability has been exploited. This approach is usually applied to unknown or undiscovered vulnerabilities (also known as zero-day vulnerabilities). Penetration tests, which can be full disclosure (white box), partial disclosure (grey box) or blind (black box) may be used to analyze the application’s response and verify the existence of the insecure signature of

potential vulnerability and possible exploitation. In some cases, it is required to have knowledge of the application's structure and perform trace analysis in order to locate the source of exploitation. In the backward approach, analysts may name and categorize newly found vulnerabilities and update related information of new vulnerabilities into existing vulnerability databases. The evaluation methods mentioned previously [5-12] could also be used to estimate the severity and impact of new vulnerabilities.

Vulnerabilities have features and attributes that make each different from the other. Based on the most typical and representative features and attributes of vulnerability, different scanning techniques have been developed to scan for and detect certain vulnerabilities. A basic vulnerability as well as its variants may be detected. A vulnerability *variant* is an instance of a basic vulnerability. A variant describes a basic vulnerability with additional details such as the location of occurrence, vulnerable parameters, or vulnerability flow (pre conditions/post conditions or transitions) etc.

1.2. Motivation

In both forward and backward approaches, since users use scanner outputs to establish vulnerability information, there is dependence on the capability of the scanner(s) used. The security solution, therefore, becomes vulnerability-specific and scanner-specific. If a user wants a more comprehensive solution that covers vulnerability variants and other vulnerabilities related to the detected one, he/she must usually refer to other resources such as external vulnerability databases, security bulletins or forums. In some cases, different scanners may

have to be used, wherein the solutions may overlap and confuse. Furthermore, vulnerability analysis requires domain expertise, which is not usually available to every organization. Another problem is that not every end user has authorized access to certain sources of vulnerability such as application source code or application server so they may not be using comprehensive vulnerability information during the remediation. The communication (connections, interactions) which exists among the software components, the applications and the operating systems can be security-relevant, and the exploitation and attack prevention on such communication may not be known by users [23]. In addition, an end user may also be provided with a vulnerability report that focuses on specific components and that lacks relevant information of an application's or system's configuration. Hunt et al. [23] commented that complex systems such as operating systems provide no easy way to distinguish between application components intermixed in file systems and configuration registries. For Web-based applications, the issues obviously require end users to have a wider view of components not only from inside the application but also from the components which are requirements for the application's operation. In such cases as described above, an approach that can account for the various vulnerability information sources, as well as provide comprehensive support for vulnerability analysis and remediation, will be an important solution.

1.3. Objectives

Given this situation, we need a method that provides the user with more comprehensive detail of vulnerability description based on available

information sources associated with the vulnerability. The method should also support forward and backward approaches and be capable of detecting and discovering vulnerability variants in vulnerability prevention. Our research analyzes a vulnerability based on its application-related properties.

1.4. Thesis organization

This report will present a survey and our own work on Web-based security and vulnerabilities analysis of Web applications. In Chapter 2, the survey of other works related to the field is presented. In Chapter 3, we present a proposal of a vulnerability analysis approach and describe the initial framework for analyzing scanner output. In this chapter, we also present the unified description method which is selected for describing vulnerability information, a proposal of fuzzy classification metrics for evaluating the scanner performance and measuring the confidence for vulnerability report, and the design of inference engine to aid vulnerability diagnostics and reports. Chapter 4 presents our proposed property-based vulnerability description model for Web applications and its application into vulnerability's impact evaluation. This model will form part of the language neutral data sub-system and will be used in the development of vulnerability analysis. Chapter 5 presents the application of the model with actual vulnerability data. Finally, we summarize our recent work and give a plan for the future work of our research in Chapter 6. The references referred to in this report can be found in the bibliography section.

CHAPTER 2

LITERATURE STUDY

In our survey, we explore the following issues, which include: (1) vulnerability analysis methodologies, (2) vulnerability description methods to provide consistent and comprehensive information that can be shared among users, (3) determination of vulnerability types and analyzing vulnerability relationships in vulnerability database and taxonomies, (4) vulnerability impact and severity.

The research approaches that are taken into consideration of this survey are summarized in Table 1, Table 2 and Table 3 below and will be presented in the following sections.

TABLE 1. VULNERABILITY ANALYSIS METHODOLOGIES

Issues	Approaches	Results
Vulnerabilities in OSs. Vulnerability variants. Abbot et al. [24]	Vulnerabilities as violations to integrity characteristics in vulnerability and the possible exploitation Using fixed and classified descriptive syntax.	Formal flaws classification. Describe many variants of integrity vulnerability
Vulnerability as flaw. Vulnerability taxonomy. Landwehr et al. [25]	Collect flaw data Classify vulnerabilities based on flaw genesis, time of occurrence during development stages and the location of flaw Operation-oriented	Generic classification. Software/hardware applicable.
Characteristic-based analysis Vulnerability variants due to difference in views and/or descriptive abstraction levels to an original vulnerability. Vulnerability taxonomy. Bishop [26, 27]	Define set of condition-based characters of vulnerabilities. Primitive conditions forming a complete characteristic set of the vulnerability. Projecting character set on specific system to create a system-dependent subset. Describe vulnerability using	Describe vulnerability with conditional character details. Vulnerability prevention by detecting and disabling certain characteristics which are common among some vulnerabilities

	<p>character subset.</p> <p>Some characters are common among several vulnerabilities.</p> <p>Vulnerabilities are categorized based on the commonality of characters</p>	
<p>Locate vulnerable components in application.</p> <p>What code patterns are most likely to occur in vulnerable components?</p> <p>Neuhaus et al. [28]</p>	<p>Mines existing vulnerability databases and version archives to map past vulnerabilities to components</p>	<p>Distribution of vulnerabilities across the entire code base.</p> <p>Feature pattern: vulnerable components shared similar sets of <i>imports</i> and <i>function calls</i>.</p> <p>Predict the vulnerabilities from features</p>
<p>How to describe the attack that exploits the vulnerability?</p> <p>Vulnerability variants</p> <p>Steffan and Schumacher [29]</p>	<p>Graph-based attack net</p> <p>Specify conditions and transitions of exploitation based on environment context and software package, software version.</p>	<p>Graph-based attack model.</p> <p>Descriptive hierarchy of attacks from context-specific to general class.</p> <p>ATiki tool.</p>
<p>Vulnerabilities present in a system are related to the set of properties that define the system.</p> <p>Vulnerability variants.</p> <p>Vulnerability taxonomy.</p> <p>Berghe et al. [30]</p>	<p>Consider key properties of vulnerability.</p> <p>Specify the correlation between vulnerabilities, application properties, application structure and existing vulnerability categories.</p> <p>Matched correlation will infer the vulnerability's existence and its properties.</p> <p>Correlation matrix</p> <p>[properties x architecture]</p> <p>[Vulnerabilities x properties]</p> <p>[vulnerabilities x categories]</p> <p>Variants are created from adjusting properties.</p>	<p>Predict the existence of vulnerability based on the application architecture, application properties with respect to vulnerabilities.</p> <p>Leverage current taxonomy.</p>
<p>Vulnerability analysis based on Constraint and Assumption perspective modeling.</p> <p>Vulnerability variants.</p> <p>Vulnerability taxonomy.</p> <p>Bazar et al. [31, 32]</p>	<p>Consider vulnerability variants as violation of constraints imposed by computer system resources or assumptions made about the usage of those resources</p> <p>Using process/Object Model of Computation.</p>	<p>Relationship between vulnerabilities, software applications, and computer system's resources.</p> <p>Vulnerability taxonomy is based on the resources (or objects) used by computational processes.</p>
<p>How individual faults can either propagate or be exploited to cause unwanted effects on systems</p> <p>Aime et al. [33]</p>	<p>Fault-tree and event-tree in a risk graph structure are used.</p>	<p>Vulnerability assessment and severity evaluation.</p>

To represent the possibilities of attack, evaluate the system and the capability of adaptation with changes Holmgren [34], Rafiei et al. [35], Swiler et al. [36], Wang et al. [37]	Vulnerability analysis using Graph-based model.	Attack graph.
Discover implicit conditional rules in a code base and to discover rule violations that indicate neglected conditions. Christodorescu et al. [38]	Behavior-based method. Mine the malicious behavior and detect malwares and malware variants	Dependence graph uniquely describes the operations performed by a given malware.
Predict vulnerability existence Woo et al. [39]	Using VDM Statistical method	Statistically project current and future vulnerabilities using VDM
Vulnerability detection: - Tainting technique (Halfond et al. [40], Huang et al. [41], and Nguyen-Tuong et al. [42]) - Fault Injection (Ghosh et al. [43]) - Penetration testing-based methods (Hurst [44])	Detect the representative features of vulnerability : Detect anomalous program states and abnormal application behaviors in response to predefined activities sent to the application intentionally.	Vulnerability detection methods
Vulnerability variants detection Scanner (Acunetix, IBM AppScan)	Change of attack parameter applied to vulnerable components of an application.	Database of attack parameters collected from experiments, known real attacks, and expert knowledge.
Vulnerability Scanner (HP WebInspect, Cenxic HailStorm)	Single possible vulnerability attack	Detect vulnerabilities

TABLE 2. VULNERABILITY DESCRIPTION

Issues	Approaches	Results
Scan results are overlapping, not complete, scanner-dependent, language-dependent, type-specific	Unify vulnerability description. XML-based: AVDL, WAS	Standard description
Databases of vulnerabilities	CVE US-CERT's NVD Scanners' databases Security Advisory boards Security forums	Collection of found/discovered vulnerabilities and related incidences. Detailed records in different formats. Specify vulnerable

	Personal descriptions	applications. Cross references to standard description
Generic resources and taxonomy	CWE	Abstract description of weaknesses (vulnerabilities)

TABLE 3. VULNERABILITY IMPACT EVALUATION AND SCORING

Issues	Approaches	Results
Measuring the impact/severity of vulnerability	Using fixed, predefined and security-related parameters Severity score calculation. Vulnerability Scoring System (CVSS) [9], Cenzic ARC-HARM score [7], nCircle Vulnerability Scoring System [5], Daily Common Vulnerability Exposure (DVE) trend over time [8]	Severity score
Heuristic evaluation of vulnerability severity	Using heuristic rating system (such as Critical, high, medium, low) Adobe Severity Rating System for Security [6], Microsoft Severity Rating System [10], Secunia “Criticality” [11], and US-CERT vulnerability severity metric [12]	Severity rating system

2.1. Vulnerability Analysis

Several tools and techniques have been developed to analyze Web-based applications vulnerability. Cova et al. [45] classify vulnerability analysis of Web-based applications according to detection models and analysis techniques (Table 4).

At a higher level, vulnerabilities may be grouped and classified based on certain features of their attributes and operations. The existing approaches allow users to understand vulnerability basics and operations generally. Abbot et al. [24] used a fixed descriptive syntax to depict the violations to integrity characteristics in a vulnerability and the possible exploitations in operating

TABLE 4. VULNERABILITY DETECTION MODELS AND ANALYSIS TECHNIQUES.

Detection model	Negative approach	Match the models of known vulnerabilities against Web-based applications to identify instances of the modeled vulnerabilities
	Positive approach	Use model of expected behavior of application to analyze the application behavior to identify abnormality caused by a security violation
Analysis technique	Static techniques	Provide set of pre-execution techniques for predicting dynamic properties of the analyzed program
	Dynamic techniques	Use a series of checks to detect vulnerabilities and prevent attacks at run-time

systems. In statements describing integrity flaw, each flaw-related element is predefined and classified into a specific class. The class value that fits the flaw scenario will be selected and used in the actual flaw description (e.g. the integrity flaw of installation illustrated in Fig. 1). This framework describes many variants of integrity vulnerability via different combinations of elements in certain interaction contexts of operating systems. The model can formally classify flaws because of the similarity of operating systems, user application

Syntax

A [Class of User] user acquires the potential to compromise the integrity of an installation via a [Class of Integrity Flaw] integrity flaw which, when used, will result in unauthorized access to a [Class of Resource] resource, which the user exploits through the method of [Category of Method] to [Category of Exploitation].

Syntax Elements

[Class of User]

- Applications
- Service
- Intruder

[Class of Integrity Flaw]

- Physical Protection
- Personnel
- Procedural
- Hardware
- Applications Software
- Operating System

[Class of Resource]

- Information
- Service
- Equipment

[Category of Method]

- Interception
- Scavenging
- Pre-emption
- Possession

[Category of Exploitation]

- Denial of Possession/Use
 - Steal equipment
 - Destroy equipment
 - Degrade service
 - Interrupt service
 - Destroy data
- Denial of Exclusive Possession/Use
 - Read/Transcribe data
 - Steal service
- Modification
 - Alter data
 - Alter equipment

Figure 1. Flaw syntax represents installation integrity flaw ([24])

programs and their security problems. Its limitation is that the resulting analyses are only textual descriptions that require expert knowledge to infer the relationship between syntax elements in order to describe actual vulnerability. The variation of descriptive syntax elements in vulnerability description also requires that different expert-dependent descriptions are involved. In our model, application-based properties are considered as descriptive “element” of vulnerability and the relationship between properties is determined by the structure of the application and its operation, hence avoiding dependency on the textual description. The variants will be presented as the combination of existing properties. In addition, our model provides scalable information describing how vulnerability becomes available and when it is exploited, what may occur. Moreover, we do not need to describe generic classes as found in the Class of Integrity Flaw and Category of Exploitation [24] because they can be inferred from the major attributes of a vulnerability and its operations.

Landwehr and his colleagues [46] analyzed program vulnerabilities using software flaws data and classified them based on flaw genesis, time of occurrence during development stages and location where the flaws occurred. Because the approach is not limited to certain classes of systems, it can be used to analyze even hardware vulnerabilities. However, the major limitation of this approach is that its descriptive methodology is operation-oriented. Thus, it does not explicitly describe application vulnerabilities caused in object property (e.g. data type in buffer overflow vulnerability or command syntax in command injection vulnerability). It also does not consider the combination of related systems in vulnerability, which is commonly seen in Web-based applications.

This classification narrows the evaluation of vulnerabilities and requires much effort to extend the analysis.

Bishop [26] classifies the vulnerabilities according to the infrastructure and network systems. Bishop comments that the definition and classification of security flaws (or vulnerabilities) are not consistent either from view of the flaw or from level of flaw conceptual abstraction. Hence, many flaw descriptions may originate from a single flaw. For example, buffer overflow can be described as incomplete validation of a parameter from the perspective of the flawed process. Under operating system perspective, this flaw is described as violable prohibition/limit because the parameter may refer to the out-of-scope address. The inconsistent parameter validation described in low level of abstraction may be referred as race condition/asynchronous-validation/inadequate-serialization problem or an exploitable logic error in higher levels of abstraction. To avoid the inconsistency and overlapping, a different model of vulnerability analysis in which vulnerability is decomposed into primitive conditions forming a set of independent characteristics of the vulnerability [27]. Each vulnerability characteristic set is a selection from the elements of a complete set of vulnerability characteristics with respect to a system. Because a vulnerability characteristic is a prerequisite condition for vulnerabilities to occur, the negation of detected characteristic will disable related vulnerabilities. The existing problems are: (a) how to determine the basic characteristic set for vulnerability and (b) the characteristics of vulnerability are not complete and unique and are not “atomic” descriptions. Thus, it is hard to develop effective tools that are capable of detecting unknown

vulnerabilities based on knowledge of existing vulnerabilities characteristics and systems such as software applications, operating systems and infrastructure hardware systems which may involve in the existence, the occurrence or the consequence of vulnerability. Moreover, defining and classifying new characteristics of unknown vulnerabilities are also desired requirements for such tools.

Neuhaus et al. [28] analyze vulnerability using software components and establish a feature pattern of vulnerable component based on the component's features. Based on the programming language syntax (C and C++) that supports import and function calls and application structure, a vulnerable component is located and its relations to other components determined during compilation and execution. The feature pattern of vulnerable components in the software is mined as reference to determine if similar imports or function calls are likely to be vulnerable. The results are applicable to structured, compile-based software where the component structure and relations are usually fixed and execution is self-contained. Thus, it is not directly applicable to describing vulnerability in Web-based applications that are interpret-based with declarative languages and script, and they have dynamic, highly interactive and context-dependent content, and are supported by several different external "components" such as Web browser, Web server, database server, embedded contents and hyperlinks.

Vulnerability can be defined as a result of weaknesses in specific software product or protocol [47]. Common Weaknesses Enumeration (CWE¹) describes

¹ <http://cwe.mitre.org/>

Figure 2. Sources of vulnerability on Multi-tier Web Application architecture with external links and user-active customization.

the weakness relationships in levels of abstraction. Moreover, vulnerability of a Web application requires particular considerations. First, Web-based vulnerability may come from sources other than the application itself. A Web application has a multi-tier architecture consisting of separately developed components: infrastructure systems, databases, processing systems (back-end tier), user interface layer (front-end tier) and Web browsers (Fig. 2). Typically, Web 2.0 has more flexible customization capability that allows end users to customize the application content with several technologies such as Asynchronous JavaScript and XML (AJAX), Flash, JavaScript Object Notation (JSON), Simple Object Access Protocol (SOAP), Representational State Transfer (REST) to facilitate communication, information sharing, interoperability, and collaboration in World Wide Web. Thus, an application

might be more vulnerable as it is exposed to more unsecured contents [48]. Secondly, next generation development techniques applied to Web-based applications create new security issues while current problems are still incompletely solved [49]. For example, while vulnerabilities induced by legacy programming techniques are not completely solved, Asynchronous JavaScript + XML (AJAX) web development technique open new attack vector for malicious code that web developer might not fully test for [50-52]. Scanners such as Acunetix has developed scanning function to scan AJAX / Web 2.0 web applications and find vulnerabilities [53].

Obviously, beside vulnerability itself, vulnerability analysis must take into account vulnerability variants in order to establish more comprehensive remediation coverage. However, vulnerability variants are considered differently from several points of view. In graph-based attack net model [29], the variants of vulnerability can be determined by specifying conditions and transitions of vulnerability attack (exploitation) based on the particular environmental context and specific software packages or versions. Bishop considers various vulnerabilities from view-based analysis on existing infrastructure systems [26]. The approach of Berghe et al. [30] assumed that a variant may come from adjusting key properties of vulnerability. A vulnerability variant is also considered as a violation of constraints imposed by computer system resources or assumptions made about the usage of those resources [31, 32]. In practice, the vulnerability variant concept has been defined differently in scanners. Variants of vulnerability can be determined as the variation of attack parameter values that can be used in one vulnerable

location of a successful attack simulation. This approach, which has been applied in Acunetix and IBM AppScan, allows a user to see any single variant of vulnerability as a sample of vulnerability attack and hence it becomes very good illustration of vulnerability exploitation. With the potential attacks inferred from these samples, user may then use an appropriate prevention. However, this method limits itself within the existing vulnerability attack dataset while the actual attack usually comes from unpredictable values. Although the scanner gives some remediation, it does not offer enough coverage for end users: they must rely on their experience to settle and extend the prevention capability such as data filtering with different values. Other scanners (e.g. HP WebInspect, Cenxic HailStorm) detect several vulnerable locations in a Web application but only suggest one possible attack value per detected vulnerable locations. The question is what will happen if the typical case generated by the scanner has already been properly filtered by application but other cases of exploitation are still undercover? The limited coverage capability of a vulnerability scanner could not simulate all possible exploitations. In other words, the vulnerability may appear not to exist to the scanner but actually it does.

Berghe et al. [30] focuses on vulnerability that occurs in certain components in Web Services. They assumed that the security vulnerability is present in the system properties. The vulnerability variants may be influenced by adjusting selected properties of existing vulnerability. A method for measuring vulnerabilities based on properties and for analyzing architectural refinement of application functional components is proposed to generate predictive

vulnerability taxonomy. The linear correlation between vulnerabilities and system properties is used to assess and weigh vulnerabilities that are likely to appear in a system component. In this approach, the new vulnerability is considered as a variation of an existing vulnerability present in a system. Like Bishop's methodology, this approach is also based on the combination of properties to determine a variant of vulnerability. The two approaches can determine the similarity or likelihood of two vulnerabilities based on property (or characteristics) set. However, as the relationships among the properties are not defined, these two approaches are limited in describing the cases in which vulnerabilities may be in relation. For example, Cross Site Scripting vulnerability may allow an attacker to generate attacker-controlled scripts to bypass the defenses of Cross Site Request Forgery vulnerability. Our property-based vulnerability model takes into consideration this issue.

Bazaz et al. [31, 32] consider relationships among software vulnerabilities, an executing process and system's existing resources. They infer a specific vulnerability concept in which a software application is vulnerable to exploits when it violates resource constraints and resource usage assumptions. The vulnerability analysis is based on system's resources that are managed by the user. However, this approach has a major problem: vulnerabilities updating requires more effort to add new constraints and assumptions from discovered and classified vulnerabilities. Moreover, many vulnerabilities, such as SQL Injection which exploits the weakness in SQL command validation, do not originate from resources constraint violation.

The recent research of Aime et al. [33] involves vulnerability assessment and severity evaluation. Fault-tree and event-tree in a risk graph structure are used to represent the insecure situation of the application, and they are used to analyze how individual faults can either propagate or be exploited to cause unwanted effects on systems. A risk graph includes a root node that represents the final negative consequence. Each possible event that can lead to the consequence becomes a child of the root. Each child, in turn, can be the root of a complete sub-tree (list) of all events which can directly lead to it. This type of graph is relevant and has been adapted to our vulnerable property relation graph. We use the graph in a broader sense: defining the relations of vulnerable property and other related properties inside the application or from/to external entities evolving from the application's operation.

Research work in [34-37] uses the attack graph approach. A possible attack state is defined as a node and an edge represents a change of state caused by a single action taken by the attacker (including normal user transitions if the attacker has gained access to a normal user's account) or actions taken by an assistant (such as the execution of a trojan horse), as a basis to determine and represent the attack and its severity impact. The fact that the number of node and edge combinations in an attack graph may explode has been alluded to in the research of Swiler et al. [36]. To solve the combinatorial explosion problem, Swiler et al. proposed an automated graph generator which receives attack templates, a configuration file, and an attacker profile as inputs and generate the attack graph.

Christodorescu et al. [38] proposed a behavior-based method to mine the malicious behavior and detect malwares and their variants. The malicious behavior is distinguished from normal behavior in interactions with the operating system and represented by a dependence graph. A dependence graph uniquely describes the operations performed by a given malware. The concept of malicious-benign behaviors approach is adapted in our approach with property-based relations, applicable to all properties that describe an application in relation to its vulnerability.

Practically, some description approaches proposed that vulnerability should be viewed from the attacker's perspective. The attack models are usually used in defense-by-attack tools in which one or several attack simulations will be launched at a target system to test the safety capability as well as the availability of security holes if exist. The ATiki tool developed by Steffan and Schumacher [29] represents the attack context using graph-based models built from attack information extracted from standard documents of development process (programming guidelines, check-lists and quality assurance requirements) and informal description databases such as Bugtraq [20, 21]. The specific context is described based on multiple-inheritance in the object-oriented paradigm. It is required that every condition and transition must be placed into context for classification and the relationship between vulnerabilities and occurrences in a concrete system must be expressible. This approach establishes a descriptive hierarchy of attacks from context-specific to general class and provides a flexible Web-based collaboration platform for cross-reference. The advantage of the proposed model is that it allows extension

to the attack description with least change to original relationships among existing attacks and their properties.

For Web-based applications particularly, there is vulnerability classification depending on error detection model and analysis technique. For Web applications, which are usually developed using high-level declarative languages, the vulnerabilities occur frequently in places where the script-based algorithms interact with the other systems or components, such as databases, file systems, operating systems, or the network [54, 55]. In other words, systems can be compromised via Web technologies, e.g. exploitation via a Web script may start a security breach. Basically, these analysis methods detect the signatures of specific features to determine known vulnerabilities available in the application. The methods presumed that vulnerability would exist at certain locations in an application and the vulnerability's representative features would be present at known related executing components. For example, methods such as tainting technique (Halfond et al. [40], Huang et al. [41] and Nguyen-Tuong et al. [42]), fault injection (Ghosh et al. [43]) or penetration testing [44] focus on detecting anomalous program states and abnormal application behaviors in response to predefined activities sent to the application intentionally. The next steps may include inferring vulnerability, tracing vulnerability origin and resources and assuming its exploitability based on known patterns of vulnerability and structure of vulnerable components and application.

Fault injection method (Ghosh et al. [43]) scans source code, inject faults to evaluate application behaviors. This method determines vulnerability as violations on application related components such as source code,

configuration, and the interactions between the components; hence, it assumes that any abnormal behaviors are considered as the existence of potential vulnerabilities. Taint tracking and analysis (e.g. Halfond et al. [40], WebSARRI tool of Huang et al. [41], Nguyen-Tuong et al. [42] or Pixy [56, 57]) monitor tainted information within data values or in information flow, analyze flow-sensitive, inter-procedural and context-sensitive data flow to automatically discover vulnerable points in a program. By focusing on the taintedness and following the propagation of tainted information, security process can filter out potential exploitation and indicate the precise original position of exploitation and provide the relevant sanitization. Both negative static and negative dynamic approach techniques use taint analysis.

Other positive and dynamic methodologies such as penetration testing [44] or attack simulation [58] use black-box testing approach. The method forces anomalous program states during executing an application and observes the application behaviors. Abnormal detection will evaluate the behavior of the application against the attack-free behavior model which is created from monitoring the application during normal operation and infer the existence of potential vulnerabilities.

While scanning techniques detect the vulnerabilities in applications, Woo et al. [39] proposed an approach where a statistical methodology works with vulnerability discovery data of detected vulnerabilities in Web browsers. Woo et al. extend the Vulnerability Discovery Model (VDM) proposed by Alhazmi and Malaiya [59] named Alhazmi-Malaiya Logistic (AML) model. Sets of vulnerability discovery data which are examined and fit to a vulnerability

discovery model of Web browser will be used for projection of both current and future vulnerability. Our approach considers a broader view of application properties which relate to vulnerability and dynamic interactions between those properties, whether they are vulnerable objects, causes or consequences of vulnerability.

2.2. Vulnerability Description

Although there are many vulnerability analysis and detection tools for Web-based applications, none of them provides a complete solution or a coverage methodology for finding vulnerabilities as desired ([45]). Different scanning techniques are implemented in Web application scanners and scanner output can help to assess the security of the Web application. However, no single scanner provides a technology-independent coverage of possible vulnerabilities. Experiments presented in [60, 61] show that the vulnerability scanning outputs are scanner-dependent, language-dependent, type-specific, and they require labor-intensive, expensive and error-prone analysis. As a result, users must rely on the combination of vulnerability detection tools and must understand different vulnerability description formats consequently. However, as shown in [60, 61], scanners usually overlap in their inferences on vulnerabilities, entailing higher costs, lower performance, data surplus and overhead in vulnerability analysis and detection. Hence, a common vulnerability description method has to be developed.

OASIS recommended using standard methodology of Web Application Vulnerability Description Language (AVDL) [62], "a standard XML format

that allows entities (such as applications, organizations, or institutes) to communicate information regarding Web application vulnerabilities”. Another is Web Application Security standard – WAS [63]. Both AVDL and WAS are projected to evolve as a common description format for every vulnerability class and help minimise the above disadvantages.

The vulnerability databases present vulnerability information in different data formats. At abstract level, CWE provides comprehensive generic description of vulnerabilities and relationships among vulnerabilities. The information is for common knowledge sharing and type determination rather than being specific to particular vulnerability or vulnerability scanner. Thus, users are usually in need of expert level skills for vulnerability analysis. The other sources of vulnerability database such as Common Vulnerabilities and Exposures (CVE²), US-CERT Vulnerability Notes Database³ provide information of specific vulnerabilities with scenario-based description and suggest references for additional data which most users are unwilling to search for. The other resources of vulnerability description come from various sources and support different types of information. For example, security vendors (such as McAfee, Acunetix, Cenzic) provide support documents of vulnerabilities that can be scanned by scanners. Software vendors (such as Microsoft Security Bulletin⁴, Adobe Security Bulletins and Advisories⁵) provide documents of particular vulnerabilities found in their products. Personal reports such as in [64, 65] describe vulnerabilities in detail of occurrence discovery in certain

² <http://cve.mitre.org/cve/>

³ <http://www.kb.cert.org/vuls>

⁴ <http://www.microsoft.com/technet/security/current.aspx>

⁵ <http://www.adobe.com/support/security/>

scenarios. In addition, security organization Websites, forums or mailing lists (such as in [18-21, 66-74]) also provide vulnerability information from different perspectives. Although the descriptions from these sources may describe the same vulnerability, they are not in common standard format and hence there is a lot of overlap. Such information is usually contributed from individual's work and therefore requires domain expertise to analyze.

2.3. Vulnerability Database and Taxonomy

The taxonomy of vulnerability aims at description, identification, naming and classification of vulnerabilities. Several approaches have been proposed for vulnerability taxonomy, many of them propose analysis methodology and provide supporting taxonomy that are specific to a certain set of vulnerabilities.

The taxonomy for program vulnerabilities proposed by Landwehr and his colleagues [25] organizes software flaws data according to flow genesis, time of occurrence during development stages and the location where the flaws occurred. This approach is passive taxonomy since it applies to the flaws that have occurred. It covers a wide range of vulnerability including hardware.

Another taxonomy proposed by Bishop [26] classifies the vulnerabilities according to the infrastructure and network systems. This taxonomy focuses on the underlying cause of the flaw rather than the description of the flaw. The taxonomy classifies vulnerability to one of six axes: the nature of the flaw, the time of introduction, the exploitation domain of the vulnerability, the effect domain, the minimum number of components needed to exploit the vulnerability and the source of the identification of the vulnerability.

A predictive taxonomy designed by Berghe et al. [30] focuses on vulnerabilities in Web services applications. A correlation matrix $[properties \times architecture]$ represents the relations between the systems and the components which describe the systems. The vulnerabilities present in the system are related to the set of properties that define the system. The representative collection of vulnerabilities and the assessment of the correlated properties will form another correlation matrix $[vulnerabilities \times properties]$. The taxonomy measures vulnerabilities against application's architecture by using the multiplication of two matrices ($[vulnerabilities \times properties] \times [properties \times architecture] = [vulnerabilities \times architecture]$). A specialization of existing vulnerability taxonomies into a vulnerability taxonomy for the system under study is formed by the following multiplication: $[architecture \times vulnerabilities] \times [vulnerabilities \times categories] \times [normalizing vector] = [architecture \times categories]$, in which, a correlation matrix $[vulnerabilities \times categories]$ is created for representing the existing taxonomy. In this approach, Berghe et al. uses linear mapping with equal weights to evaluate the likelihood of vulnerabilities. The likelihood measure provides a reliable heuristic for vulnerability detection. This methodology requires that the selection of properties should be complete despite the fact that information of vulnerability may be missing or data of vulnerability description are usually complicated. Moreover, the selection of a minimal set of properties from available set is also difficult. Besides, since the properties contribute differently to certain vulnerabilities, alternative adaptive weighing schemes may be needed.

Bazaz and Arthur [32] establish a taxonomy of vulnerability using the Process/Object Model of Computation. Vulnerabilities are classified within the taxonomy according to the categories and subcategories identified by a computer system's resources. This approach also considers the formal relationship among vulnerabilities.

Considering SQL injection attacks (SQLIAs) and relative information collected from various sources, Halfond et al. [75] provides a comprehensive survey on different types of SQL injections to support evaluation of detection and prevention techniques. SQLIAs can be categorized by two characteristics: injection mechanisms and attack intents. The definition and evaluation provide a more descriptive, concise and common format of information of SQL injection vulnerabilities: name, attack intent(s), attack description, attack example, and a set of references to documents related to the attack in which the attack variations are also included. However, it is clear that vulnerability analysis still needs the combination of several tools in order to detect the potential vulnerabilities in the application.

The CWE structure description of vulnerabilities provides generic description used among experts and end users. The description, however, cannot support automated analysis at any level and require manual effort for specific data and information.

2.4. Vulnerability Impact Evaluation and Scoring

Another issue is that the security impact of vulnerability should be evaluated. Current statistical evaluation and scoring methods (CVSS [9], Hailstorm [76],

nCircle[5]) use selected fixed, predefined and security-related parameters in evaluating impact and calculating severity score (impact score) of vulnerability. These approaches have two advantages. Firstly, every vulnerability will be evaluated on the same set of security-related parameters and hence, they are evenly examined and compared. Secondly, evaluation only focuses on fixed parameters with predefined set of possible values. Those parameters can be easily recognized, and commonly used among different levels of users. Moreover, statistical methodology can be applied to infer the evaluation result. Thus, the evaluation results (generic and detailed reports) are able to show the severity of vulnerability and support severity comparison among vulnerabilities. Additional metrics such as temporal metric proposed by CVSS [9] or Daily Vulnerability Exposure based (DVE_{date}) with Vendor Time To Fix (TTF_v) [8] used to measure the existing capability of detecting vulnerability or environmental metric (in [9]) are simple heuristics and they can be easily calculated by any user.

However, the above approaches have limitations. It becomes more common that users view a vulnerability differently according to their perspectives and want to consider its impact in particular scenarios. Specifically, it can be seen that in real scenarios the relationship of vulnerability components (application property-based) and their attributes may vary the exploitations of vulnerability and alter the effect and consequence. For example, real-time (or near real-time) Web applications may lose their timeliness due to unpredictable delay in processing or may face service blocking due to overwhelming requests (denial-of-service attack). The existing methods usually separate the core and context-

based situations (such as CVSS), or only focus on certain set of parameters. The problem is that, with those methods, the user may base his security evaluation on expert measurements which would be rather fixed and tend to be generalized among several different systems while each system may already be certified securely at certain levels. Typically, if the evaluation results in remediation that is insufficient to deal with the actual situation, the system will be exposed to even more vulnerabilities. Our model has built-in impact factors and automatically provides severity estimation.

Our research suggested that vulnerability should be described based on the following aspects:

- The description must rely on a set of commonly known properties. In our research, we concentrate on application property set. Although the view of each factor (application developer, application administrator and end user) is at different levels of understanding, the properties that describe fundamental characteristics of application are standard.
- The fact that property attributes and property relationship can be known to users calls for a common description.

In the following sections, we will present the design of our proposed vulnerability analysis system, the experiments on selection of unified description language, and the basic concepts of property-based analysis model in Web vulnerability analysis. Based on the proposed theory, a mathematical evaluation of vulnerability will be described.

CHAPTER 3

WEB APPLICATION VULNERABILITY ANALYSIS FRAMEWORK

Many vulnerabilities are common with different Web languages, the rest are specific to language characteristics, compile process, and executing methodology. The question is how to detect vulnerabilities with least dependence on development technologies and programming languages. We proposed a new framework for Web-based vulnerability analysis (Fig. 3) ([77]). Our approach is to develop an analysis technique that would cover vulnerabilities independent of Web technologies.

In this framework, scanner results will be inputs to the analysis system (see Fig. 3). At the current stage of development, the document filter works with HTML, XML and PDF format output files. This filter is integrated in a Web language-neutral data sub-system. This front-end component serves as the data interface to the rule-based inference engine (Fig. 3).

Figure 3. The framework for Web application vulnerability analysis.

Rule base contains inference rules established from standard description of vulnerabilities. Deepak et al. [78, 79] proposed fuzzy classification metrics that are used to grade web application scanners and vulnerabilities so that scanner performance can be evaluated and confidence levels can be computed for vulnerability reports. The new scanner evaluation parameters are derived from a set of five assertions and the initial fuzzy scanner metrics. Based on that, the 1st and 2nd degree confidence reports are calculated and used to provide information aiding the initiation of suitable remediation steps to be taken. In our system, the rule-based inference engine uses the rule base to analyze data extracted from scanner outputs and provides report and diagnostics [80, 81]. Information in the language neutral data sub-system, vulnerability reports, and diagnostics are described in XML-based AVDL style format ([60, 61]).

In Chapter 4, the design of the property-based vulnerability description model is described. This model will form part of the language neutral data sub-system and will be used in the development of vulnerability analysis and impact estimation (vulnerability scoring sub-system).

CHAPTER 4

PROPERTY-BASED VULNERABILITY DESCRIPTION MODEL FOR WEB APPLICATIONS

We propose a graph-based model that uses application properties and their relationships to describe vulnerability. We first present the definition of application properties that will be used in our vulnerability description model. Then, a new graph-based model called *vulnerable property relation graph* is presented. The design of this model focuses on vulnerability analysis and description problems. The secure state attribute of property is also defined. Finally, the vulnerability impact evaluation and vulnerability impact calculation based on VPRG are presented [82, 83].

4.1. Application Properties in Vulnerability Analysis

In our research, we specify application's properties which can be used by a user (usually an attacker) to exploit the vulnerable state [26] of an application. A *vulnerable property* of an application is any application's property which enables a user to obtain and/or modify information without authorization, to grant or deny access to a resource without authorization, or to perform other unauthorized operation(s) that cause the application to work improperly or crash. For Web applications, a vulnerable property may allow a user to break the integrity of the application or violate the operation of other application(s), or disclose data which can be used by a user to analyze and violate data privacy.

From a vulnerable property in a Web application, we can determine cause and consequence properties from which users can describe the vulnerability,

infer the attributes of possible attack (such as attack conditions, attack flow) and determine the possible exploitation process and exploitation consequence of a vulnerability based on its properties and property-property relations.

Considering the scope of a typical vulnerability, a Web application W can be described by a tuple of four property sets which are pre-condition, behavior, entity and communication:

$$\begin{aligned} W &= (precondition, behavior, entity, communication) \\ &= (\{p^{pcond}\}, \{p^{behavior}\}, \{p^{entity}\}, \{p^{comm}\}) \end{aligned}$$

Definition 1 (Pre-condition). A *pre-condition* property is a conditional predicate that determines the scope of working requirements and environment and describes the operational conditions including assumptions, constraints, and initial states/conditions needed for the application to operate properly.

We denote a pre-condition property as a predicate:

$$p_i^{pcond} = g(t),$$

where t is a property of the application and g is a predicate that represents the pre-condition related to that property. For example, the operational environment requirement of a Web application is that the Web browser must have JavaScript setting enabled, which can be represented by:

$$p_i^{pcond} = JavaScriptEnable(WebBrowser).$$

An application property may require several pre-conditions to hold true in order for it to exist or operate properly. To represent such complex pre-conditions, we use combination of predicates. For example, a Web page in a

Web application may require a JavaScript-enabled Web browser for client-side processing and dynamic form fields that validate and receive only standard alphanumeric text, which can be represented as follows:

$$\begin{aligned} &JavaScriptEnable(WebBrowser) \\ &\wedge TextStandardAlphanumericValidate(FormField) \end{aligned}$$

In our prototype model, we restrict the logical operations on a predicate combination to conjunction $\wedge$ and disjunction $\vee$. In practice, pre-conditions may be defined in requirement specifications and in the design phase of a software development lifecycle. They are either implemented as default application parameters or set as operational conditions for the working application. Because violation of any pre-condition may cause vulnerability to an application, we consider a pre-condition as a *cause*- part of a vulnerability of certain type. In vulnerability detection, a pre-condition set presents potential vulnerability availability corresponding to the realization of weak condition or condition violation.

Definition 2 (Behavior). *Behavior* is the operational functionality of an application. A behavior is defined as a procedure with in-arguments, out-arguments and a list of other behaviors invoked by this behavior:

$$p_i^{behavior} = f(< in_{arg} >, < out_{arg} >, < p_{j \neq i}^{behavior} >)$$

The input arguments $< in_{arg} >$ and output arguments $< out_{arg} >$ are defined by data types and described as data or entity properties.

The invocation relation between behaviors provides a way to represent the function-based relationship between correlated behaviors. In Web applications, a behavior may be an operation of interpreted codes (such as script codes or page markup codes) which are inserted into the Web page without arguments (null argument behavior). Such codes are interpreted and executed on-line. In this case, behaviors will be described by the direct relations between current behavior with precedent behaviors and/or a list of its follow-up behaviors, if any. We use a behavior correlation graph to represent the activities of the application on the basis of invoked-by/precedent and invoke/followed-by relations. The behavior correlation graph reflects the logical steps of a potential attack (or series of attacks) and its consequence behind a sequence of application activities during vulnerability exploitation. We define the behavior correlation graph and the relations between behaviors as follows:

Definition 3 (Behavior correlation graph). A behavior correlation graph $BG = (B, E)$ is a connected directed graph, where the node set B is a set of behaviors and for each pair of nodes $n_1, n_2 \in B$, there is an edge from n_1 to n_2 in E if and only if n_1 is followed by n_2 .

EXAMPLE: Suppose we have a Web application that reads an employee ID entered by a user, processes a search query at the server and displays the search result at the user's Web browser. The activity sequence for this function has the following behaviors:

$$\begin{aligned} n_1 &= \text{interpretWebpage}(\text{page_code}), \\ n_2 &= \text{displayWebpage}(n_1), \\ n_3 &= \text{sendRequestToServer}(\text{request_string}), \end{aligned}$$

Figure 4. Behavior Correlation Graph.

$n_4 = \text{validateStandardAlphanumeric}(\text{FormField}, \text{employee_ID}),$
 $n_5 = \text{search}(\text{employee_ID}),$
 $n_6 = \text{sendResponseToBrowser}(\text{page_code}, \text{response_string}),$
 $n_7 = \text{interpretWebpage}(\text{page_code}, \text{response_string})$
 $n_8 = \text{displayWebpage}(n_6).$

We also denote edge e_{12} from node n_1 to n_2 , e_{23} from node n_2 to n_3 , and so on. The corresponding behavior correlation graph $BG = (\{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}, \{e_{12}, e_{23}, e_{34}, e_{45}, e_{56}, e_{67}, e_{78}\})$ is shown in Fig. 4 ■

A behavior can be invoked by or invoke other behavior(s). To determine the behavioral relationships of a behavior to other behavior(s), we have:

Definition 4 (Behavior Precedent graph). A behavior precedent relation graph $\text{precedent}(n, BG)$ is the sub-graph $BG' = (B', E')$ of BG that contains every node n' that makes a call to n or must be manifested before n .

EXAMPLE: From Fig. 4, the behavior precedent sub-graph of n_3 is $\text{precedent}(n_3, BG) = (\{n_1, n_2\}, \{e_{12}, e_{23}\})$ ■

Definition 5 (Behavior Succedent graph). A behavior succedent relation graph $\text{succedent}(n, BG)$ is the sub-graph $BG' = (B', E')$ of BG that contains every node n' that is called by n or manifests after n .

EXAMPLE: From Fig. 4, the behavior succedent sub-graph of n_3 is $succedent(n, BG) = (\{n_4, n_5, n_6, n_7, n_8\}, \{e_{34}, e_{45}, e_{56}, e_{67}, e_{78}\})$ ■

The precedent relation graph is used to describe all behaviors that must execute in precedence, directly or indirectly, to a vulnerability-related behavior. The succedent relation graph describes all application behaviors that are direct or indirect consequences of a vulnerability-related behavior.

Definition 6 (Entity). Entity or data-related properties are objects with their own attributes evolving during the application's operation.

In our model, the entity concept refers to the application's internal and external objects of several types. The internal objects may be Web-based components (e.g. Web pages, script code files, cookies), variables, data records. The external objects may include embedded Web-based components from other Web applications, remote procedures or embedded plug-ins/add-ins, Web server systems, Web browser, database management systems. To describe an entity, we define an abstract entity type and entity as an instance of entity type.

Definition 7 (Entity Type). Entity type $T = (id, category)$ defines the type of entity based on the software category. An entity is an instance of entity type:

$$entity \xi: (A, S) \rightarrow T$$

$$p_i^{entity} := \xi(\{a_j\}, s) = t, t \in T, s = \{in, ex\} \in S$$

where A is the attribute set used to define entities in an application and S is the scope attribute set of an entity with in and ex indicating whether the entity is internal or external to the application scope, respectively.

The relationship of entities is determined in the entity relationship model (ERM). In Web-based vulnerability, the entity concept is used not only for determining the vulnerable entity inside the Web application but also the entities outside the application and evolving in the operation of the Web application and related to the vulnerable property.

Definition 8 (Vulnerability-related Entity Set). The vulnerability-related entity set E_v^W is a set of vulnerable entities of Web application W and entities outside W which may contain vulnerabilities and influence the secure state of W .

EXAMPLE: While scanning a Web application, the scanner detects a vulnerability in its Web page (*file.html*) and also detects that the Web server *WebServer* on which the application runs is vulnerable due to non-upgrade to secured version or operation without relevant security patch. We may express this as:

$$E_v^W = \{(file.html, in), (WebServer, ex)\}$$

In this set, *file.html* and *WebServer* are of *webpage* type and *system* type, respectively ■

Definition 9 (Communication). A *communication* property is a property that determines input/output method, protocols, data exchange formats and data encoding methods. We denote a communication property as a sequence of behavior(s) work with certain data (input) and producing output:

$$p_i^{comm} = f(< in_{arg} >, < out_{arg} >, < behaviorSeq >),$$

where $|in_{arg}| \neq 0$ and $|out_{arg}| \neq 0$ and $\langle behaviorSeq \rangle$ represents a sequence of behavior(s) involved in the processing of input data to output.

EXAMPLE: HTTP Protocol Stack (HTTPSTK) in Novell eDirectory⁶ with a vulnerable HTML code filter can be expressed as:

$$p^{comm} = HTTPSTK (UserInput, DisplayOutput, (proc_i(in, out))),$$

where $(proc_i(in, out)), i = 1..n$ defines the sequence of n processing steps executed by the vulnerable HTML code filter ■

Definition 10 (Communication property relation). The relation of communication properties may be defined as *communication exchange* relation $exch(comm_1, comm_2)$, where communication property $comm_1$ fulfills data exchange specifications with communication property $comm_2$.

EXAMPLE: “Cross-protocol scripting” vulnerability [65, 84-86] allows an attacker to send multi-part POST submission of a malicious payload in HTTP protocol to a vulnerable non-HTTP services. ■

4.2. Describing Vulnerability Using Vulnerable Property Relation

Graph (VPRG)

In this section, we introduce a property-based graph to describe vulnerability in a Web application. We first define property-property relationships. Then, we define the graph that describes the vulnerability using vulnerable property and its relationships with other application properties.

⁶ <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-5093>

4.2.1. Property-Property Dependencies

In the previous section, the relationships between precondition to precondition, entity to entity, behavior to behavior, and communication to communication properties have been defined. A pre-condition property is a cause of the other properties. We can infer the relationship among properties from the logic structure of the application. Here, we define the relationships among properties of different types:

REMARK (*Behavior – Communication relationship*). Behavior is a generalization of communication property. Definition 11 can be derived from Definition 2 and Definition 5.

Definition 11 (*Entity – Behavior relationship*). Given an entity property, p^{entity} , and a behavior property, $p^{behavior}$, in application W , a relationship between p^{entity} and $p^{behavior}$ exists if:

- p^{entity} is input of $p^{behavior}$ or
- p^{entity} is output of $p^{behavior}$ or
- $p^{behavior}$ is a functional behavior of p^{entity} . In this case, $p^{behavior}$ is called by p^{entity} . For example, the script execution of Web browser can be described as: $webBrowser \rightarrow scriptExecution()$
- p^{entity} is activated by $p^{behavior}$. For example, to describe the execution of SQL-based script code that would activate and have effect on the database, we may exemplify as follows: $scriptExecution(ScriptCode) \rightarrow database$.

REMARK (*Entity – Communication relationship*). From definitions 9 and 12, a communication property takes data entities as input and produce data entities as output. Moreover, two entity properties that are communicating must include at least one intermediate communication property.

For example, GET method is considered as a communication property that sends data as part of request URL from client side Web browser to Web server, described as follows:

WebBrowser

$\rightarrow GET((ACTION, urlParameter), (ACTION, urlParameter),$

$sendRequest()) \rightarrow WebServer$

REMARK (*Decomposition of property*). A property can be decomposed into simpler properties without changing its entry and exit connections.

Taking the above example, to consider request data in details, we may decompose $GET()$ as follows:

$WebBrowser \rightarrow \begin{matrix} \nearrow \\ \searrow \end{matrix} \begin{matrix} ACTION \\ urlParameter \end{matrix} \begin{matrix} \searrow \\ \nearrow \end{matrix} \rightarrow$

$GET((ACTION, urlParameter), (ACTION, urlParameter), sendRequest())$

$\begin{matrix} \nearrow \\ \searrow \end{matrix} \begin{matrix} ACTION \\ urlParameter \end{matrix} \begin{matrix} \searrow \\ \nearrow \end{matrix} \rightarrow WebServer$

The $\begin{matrix} \nearrow \\ \searrow \end{matrix}$ and $\begin{matrix} \searrow \\ \nearrow \end{matrix}$ denote the branches of decomposition. In this case, user may distinguish between *ACTION* and *urlParameter* from $GET()$ and consider two properties: the *ACTION* that determines how a request message is received and the *urlParameter*, which is part of the communication message. In fact,

the exploitation of Web Server vulnerability (e.g. cross site scripting) may be caused by an alternation of *urlParameter* not at *GET()*. Hence, *urlParameter* should be considered a cause and *GET()* as a behavior in a path to the vulnerable property.

4.2.2. Vulnerable Property Relation Graph (VPRG)

We will now use the application properties including vulnerable property and its related properties and property-property relationships to describe vulnerability. In application *W*, let p_v denote a weak and exploitable property which is subject to vulnerability v . From now, we call p_v a vulnerable property. An exploitation of vulnerability will be represented as the effect to vulnerable property and its related properties.

Definition 12 (Vulnerable Property Relation Graph). Vulnerable property relation graph (VPRG) $G_v := (P, E, \mathcal{R})$ is a weighted directed graph that describes positively detected vulnerability v and represents the relationship of v -related vulnerable property with other properties in the application. In G_v , each node $p_i \in P$ represents a property of the application. A directed edge:

$$e_{ij} \in E, e_{ij}: p_i \xrightarrow{w_{ij}} p_j$$

represents the ordered relation of two properties p_i and p_j in which p_i is precedent to p_j . Weight $w_{ij} \in \mathcal{R}$ is assigned to each edge e_{ij} to measure the strength of the relationship from p_i to p_j .

Figure 5. VPRG of vulnerability at p_v .

The VPRG of vulnerability v consists of one root node which is the vulnerable property (e.g. node p_v in Fig. 5) and other nodes that are properties directly or indirectly related to the vulnerable property. Generally, VPRG is a reduced model of the application structure - only the vulnerable property and its related properties are shown.

In VPRG, edge $e_{ij}: p_i \xrightarrow{w_{ij}} p_j, j \neq i$ is called *cause-consequence* relation. Property p_j is said to be passively involved (passive node) and receives consequence caused by p_i and p_i is active property (active node) and passes consequence to p_j . Moreover, we distinguish between the *cause*- sub-graph and *consequence*- sub-graph of vulnerability in its VPRG as follows.

Definition 13 (Cause graph). A *cause*- graph $\text{cause}(v, G_v)$ is the sub-graph $G'_v := (P', E', \mathcal{R}')$ of G_v that represents the possible cause(s) of vulnerability v , where $P' \subset P$, $E' \subset E$ and $\mathcal{R}' \subset \mathcal{R}$. Possible causes of vulnerability v are defined by the relationship between vulnerable property p_v and its precedent properties:

- $\forall p' \in P'$, there is path from p' to p_v .

Definition 14 (consequence graph). A consequence- graph $consequence(v, G_v)$ is the sub-graph $G_v'' := (P'', E'', \mathcal{R}'')$ of G_v that represents the possible consequence of vulnerability v , where $P'' \subset P$, $E'' \subset E$ and $\mathcal{R}'' \subset \mathcal{R}$. Possible consequences of vulnerability v are defined by the relationship between vulnerable property p_v and its followed-by properties:

- $\forall p'' \in P''$, there is path from p_v to p'' .

For example, in Fig. 5, the cause- and consequence- graph of vulnerability v are:

$$\begin{aligned} cause(v, G_v) &= (\{p_{v.0}\}, \{e_{v.0,v}\}, \{w_{v.0,v}\}), \\ consequence(v, G_v) &= (\{p_{v.1}, p_{v.2}, p_{v.3}, p_{v.4}\}, \\ &\quad \{e_{v,v.1}, e_{v.1,v.2}, e_{v.1,v.3}, e_{v.3,v.1}, e_{v.3,v.4}\}, \\ &\quad \{w_{v,v.1}, w_{v.1,v.2}, w_{v.1,v.3}, w_{v.3,v.1}, w_{v.3,v.4}\}) \end{aligned}$$

In VPRG, circular connection is used to present the relationship of two properties interacting with and causing consequence to each other. For example, p_i precedes p_j and operates before p_j , then p_j runs and in turn passes the operation back to p_i . The relationship between $p_{v.1}$ and $p_{v.3}$ in Fig. 5 illustrates a circular connection in the VPRG. Moreover, this circular connection can be extended to more than two properties. Weights of edges between two adjacent properties can be different: $w_{ij} \neq w_{ji}$.

The vulnerable property relation graph can be used (1) as a defense-in-depth measure to protect against any possible new vulnerability vectors identified and (2) in estimating the consequences of vulnerability and scoring the severity of vulnerability based on the spread of consequence to properties in relation with

the vulnerability. In the following subsections, we will define the consequence of vulnerability exploitation based on the difference of property states.

4.2.3. Security-related Operation State of Property

An application property p has a pre-defined finite set of operation states S_p determined at requirement specification and design stages in the software development life cycle: $S_p = \{s_{p,i}, i = 1, \dots, n\}$, where n is the number of states currently associated with p . It is expected that proper operation states of the property p is within S_p . Operation of application causes an involved property to change its state.

In application security, if vulnerability v exists in property p , the exploitation of v will cause p to change its state from $s_{p,i} \rightarrow s_{p,j}$, $s_{p,i} \in S_p$. The result state $s_{p,j}$ is the expected state needed for exploitation and would be either:

- A predefined state of p : $s_{p,j} \in S_p$. For example, attacker exploits XSS vulnerability to gain valid user access privilege (a predefined state of access privilege) in order to post unregulated material to a trusted Web site for other valid users.
- An unknown or undefined state of p : $s_{p,j} \notin S_p$. For example, exploitation of buffer overflow vulnerability causes a running application to enter an unspecified state of behavior. On the other hand, in many attacks that exploit buffer overflow vulnerability, the after-attack behavior of the application is known to and prepared for by the attacker, but unknown to an application user.

In general, it can be inferred that a property that relates to the vulnerable property would change its state accordingly due to the exploitation. We call the change of a property's state in security exploitation the *security state transformation*. Therefore, the change of state of properties involved in vulnerability represents the effect of exploitation on an application. We will use the property's operation states to determine the impact of vulnerability exploitation.

For certain vulnerability exploitation, we may not know exactly in advance whether the proper resultant state of a vulnerability's property is in its pre-defined set of operation state. In such situation, we can assume and include an abstract *unsecured* state of the involved property. The *unsecured* concept defines the lowest level of the security and its actual meaning depends on the type of property and the situation of vulnerability in which the property is involved. For example, in the buffer overflow attack, the memory buffer property may have "*unsecured*" state applied for all situations in which the exploitation can access to memory areas beyond buffer size.

4.2.4. Matrix Form of VPRG – Vulnerable Property Relation Matrix (VPRM)

We represent VPRG G_v using an adjacent matrix– Vulnerable Property Relationship Matrix – VPRM.

The VPRM is a matrix $A_v(n \times n)$ that satisfies the following specifications:

- columns and rows represent the properties $p_i \in P$ in G_v ,
- first index of matrix column and row represent the vulnerable

property p_v ,

- $A_v(i, i) = 0$; $A_v(i, j) = w_{ij}$, $i \neq j$ represents *cause-consequence* relation between p_i and p_j : $p_i \xrightarrow{w_{ij}} p_j$, otherwise $A_v(i, j) = 0$.

The first row of VPRM represents the direct consequence from vulnerable property to other properties. The non-zero entries in the first row and the following rows which have index values equal to the column index of non-zero entry represent the propagation of vulnerability consequence, i.e. if a non-zero value is found, the calculation is done for the row whose index value corresponds to the column in which the non-zero value appears. Likewise, the first column represents the direct cause of properties precedent to the vulnerable property. The non-zero entries in the first column and the following columns which have index values equal to the row index of non-zero entry will represent the trace of causes of vulnerability. For example, VPRM shown in Table 5 represents the VPRG in Fig. 5 (p_v is the vulnerable property). The algorithms for tracing consequence and cause in VPRM are defined in the next section.

TABLE 5. VPRM REPRESENTS VPRG

A_v	(1)	(2)	(3)	(4)	(5)	(6)
p_v (1)	0	0	$w_{v,v.1}$	0	0	0
$p_{v.0}$ (2)	$w_{v.0,v}$	0	0	0	0	0
$p_{v.1}$ (3)	0	0	0	$w_{v.1,v.2}$	$w_{v.1,v.3}$	0
$p_{v.2}$ (4)	0	0	0	0	0	0
$p_{v.3}$ (5)	0	0	$w_{v.3,v.1}$	0	0	$w_{v.3,v.4}$
$p_{v.4}$ (6)	0	0	0	0	0	0

4.3. Impact Evaluation Using VPRG

We distinguish and evaluate the impact of cause and consequence of vulnerable property differently. While the analysis on cause focuses on how related properties are combined in order to express availability and exploitability of vulnerability, consequence analysis uses the connection between properties to evaluate the distribution of impact of that vulnerability. We will present the factors that influence the impact evaluation of cause-properties and consequence-properties in this section.

4.3.1. Combination Relations of Cause-Properties

The impact of cause-properties in combination would be considered differently with conjunction and disjunction. Basically, conjunction of cause-properties (e.g. $(p_{i-2} \wedge p_{i-3}) \rightarrow p_{i-1}$ in Fig. 7) requires the availability of every component properties to cause a certain impact to their consequence-property, and thus the impact of a property in conjunction depends on the availability of the others. As a result, the conjunction between cause-properties will restrict the availability of vulnerability.

In VPRG graph, if p_i and p_j are conjunctively related and lead to p_k , we denote the conjunction ($\wedge$) of properties p_i and p_j as follows:

$$(p_i \wedge p_j) \rightarrow p_k$$

Conjunction is compound, which means that its component properties depend on each other. The conjunction is considered as single node which consists of contribution of component properties and has compound effect on

Figure 6. Conjunction (a) and disjunction (b) of properties in VPRG graph.

the consequent property. In VPRG diagram, we use a short bold line to represent the conjunction of properties (Fig. 6a). The bold line gathers all components' connection of conjunction and will connect to the subsequent property by a single connection.

On the other hand, disjunction of cause-properties only requires the availability of at least one property to cause a certain impact to the consequence-property. Hence, the disjunction between cause-properties leads to the higher availability of vulnerability.

In VPRG graph, if p_i and p_j are disjunctively related and lead to p_k , we denote the disjunction ($\vee$) of properties p_i and p_j as follows:

$$(p_i \vee p_j) \rightarrow p_k$$

Disjunction of properties is distributive, which means that in the disjunction a property will cause impact to the consequence-property without depending on the other properties:

$$(p_i \vee p_j) \rightarrow p_k \equiv (p_i \rightarrow p_k) \vee (p_j \rightarrow p_k)$$

In VPRG diagram, we denote the disjunction of properties as connections from components of disjunction to the consequence-property (Fig. 6b). Later,

we define the cause-chain concept of cause-properties which will be used to determine the core structure of the cause of the vulnerability.

Definition 15 (Cause-chain). A chain of cause-properties, *cause-chain* C , is a sub graph of a cause- graph G' in which the properties are consequently related (chain) and leads to the vulnerable property, p_v . Alternatively, a cause-chain is a *simple path* from a node $p_i \in G'$ to p_v . We define a cause-chain C as follows:

$$C \subset G': \{p_1 = p_i, p_2, \dots, p_k \text{ such that } p_j \rightarrow p_{j+1} \text{ for all } j, 1 \leq j < k \\ \text{and } p_k \rightarrow p_v\}$$

A cause-chain has the following characteristics:

- A cause-chain may begin at the farthest node from p_v in the cause-graph and ends at the node that presents direct cause on the vulnerable property.
For example, there are two cause-chains in Fig. 7: $(p_{i-2} \wedge p_{i-3}) \rightarrow p_{i-1} \rightarrow p_i \rightarrow$ and $p_j \rightarrow$ (note that the dashed nodes p_k , p_{j-1} and p_{i-4} represents the additional nodes to be considered later).
- If a conjunction exists in a cause-chain, the conjunctive nodes and their cause sub-graph constitute part of that cause-chain e.g., in Fig. 7 $(p_{i-2} \wedge p_{i-3})$ in cause-chain $(p_{i-2} \wedge p_{i-3}) \rightarrow p_{i-1} \rightarrow p_i \rightarrow$.

Figure 7. Cause graph of vulnerable property p_v

- Disjunctive properties belong to different cause-chains (*branch* chains).

For example, in Fig. 7 p_i belongs to $(p_{i-2} \wedge p_{i-3}) \rightarrow p_{i-1} \rightarrow p_i \rightarrow$ and p_j belongs to $p_j \rightarrow$. Hence, there may be more than one cause-chain in a cause-graph.

Definition 16 (Basic Cause-chain). A basic cause-chain of a cause-graph is the longest chain of cause-properties that has no disjunctive branch.

In vulnerability description, the basic cause-chain represents the *core* descriptive structure of the cause part of vulnerability: the properties in a basic cause-chain will assure availability of vulnerability at the vulnerable property. In addition, the structure of a basic cause-chain does not change if a new cause-property is added to the cause-graph. For example, in Fig. 7, $(p_{i-2} \wedge p_{i-3}) \rightarrow p_{i-1} \rightarrow p_i \rightarrow$ is a basic cause-chain of vulnerability v at p_v . If p_k is added as shown, there will be two precedents to p_i and the first chain will no longer be basic as a disjunctive branch has been introduced. In this case, $p_i \rightarrow$ becomes the new basic cause-chain.

Cause-chains, basic cause-chains and cause properties have the following characteristics:

- A basic cause-chain always exists in a VPRG. Hence, there is always at least one cause-chain.
- A cause-chain that comprises of a basic cause-chain and additional properties denotes alternative cause(s) of the core. We call these alternative cause(s) of the core, *variants* of vulnerability.
- Addition of a new cause-property precedent to the farthest node from p_v

in a cause-chain will create a variant. For example, in XSS vulnerability, the basic cause-chain would be $(user_controllable_input(E) \wedge input_validation_condition(PC)) \rightarrow$, where E, PC are abbreviations of entity and pre-condition, respectively. Further analysis may determine that the $user_controllable_input(E)$ comes from a source such as $HTTP_request_message(E)$, which is then presented in VPRG as $HTTP_request_message(E) \rightarrow user_controllable_input(E) \rightarrow$. In Fig. 7, vulnerability v at vulnerable property p_v would be caused by violation at property p_i advancing to it or by violation at p_{i-1} precedent to p_i . In the attacker's view, the basic cause-chain $p_i \rightarrow$ may be considered as basic information for an exploitation; attacker can exploit the weakness at p_v with information from cause-property, p_i . With $p_{i-1} \rightarrow p_i \rightarrow p_v$, the attacker has more information from cause-property p_{i-1} to perform exploitation at p_v . Hence, the possible exploitation in this case may be considered as an alternative (variant) method of exploitation of v .

- Addition of new cause-property as a disjunctive node to an existing node in a cause-chain will also create a variant of vulnerability. In the case of the XSS vulnerability example above, additional source of input data $database_field_value(E)$ in the cause part of vulnerability is presented as $(HTTP_request_message(E) \vee database_field_value(E)) \rightarrow user_controllable_input(E) \rightarrow$. This comprises two variants of XSS vulnerability: reflected XSS and stored XSS. In Fig. 7, p_k would provide additional information for exploitation of v at p_v via $p_k \rightarrow p_i$, creating a new variant.

- Addition of new cause-property as a conjunctive node to existing node(s) will not create variant of vulnerability but add more conditions to the existing exploitation of vulnerability. In Fig. 7, p_{i-4} is an additional requirement to the availability of vulnerability v and would need to be considered by an attacker to exploit v with steps denoted by chain $(p_{i-2} \wedge p_{i-3} \wedge p_{i-4}) \rightarrow p_{i-1} \rightarrow p_i \rightarrow$.
- If a precedent cause-property is added at the farthest node from p_v or in conjunction with existing node(s), the basic chain will be extended (e.g. in Fig. 7, node p_{j-1} will extend the basic chain $p_{j-1} \rightarrow p_j \rightarrow$ and node p_{i-4} will extend the basic chain $(p_{i-2} \wedge p_{i-3} \wedge p_{i-4}) \rightarrow p_{i-1} \rightarrow p_i \rightarrow$).

4.3.2. Distribution of Consequence and Consequence-Property Cardinality

The impact of vulnerability v in a consequence evaluation, on the other hand, needs no consideration about the combination between consequence-properties. It will be counted as the impact of v on every consequence-property of vulnerable property p_v . However, we need to consider the connections of discovered consequence-properties in order to evaluate the distribution of vulnerability impact. In addition, a consequence-property may have more than one instance and the actual impact of vulnerability may multiply to every existing instance(s) of property, thus we suggested that the estimated number of property's instance is also involved in impact evaluation. For example, SQL injection vulnerability would allow the attacker to display customer data record of a company which can be described as the consequence behavior property *display_Customer_record(DB, customer_ID)* with cardinality: $0..n$. The

range $min \dots max$ of property cardinality value depends on the actual number of property's instances (e.g., software vendors would know exactly the number of customer records are influenced from SQL injection vulnerability ($n \geq 1$) while end user only considers his own record ($n = 1$) has been used illegally after exploitation). In certain situations, the maximum value is unknown or infinite and will be denoted as $*$. The minimum value is normally set to 1 indicating that at least one instance of according property has been affected. However, it can be set to 0 if the analysis suggests the possible affection of vulnerability on indicated property but has no actual confirmed information about the affected property.

4.3.3. Impact Estimation of Vulnerability Exploitation

4.3.3.1. Impact Estimation

In section 2, we have discussed security-related operation state and security state transformation of application properties. In this section, we apply these two concepts to estimate the severity of vulnerability exploitation by calculating the impact score for vulnerability described based on the property-based description model.

Let S_p be the set of operation states associated with property p . An activity t that occurs on property p and causes it to change state from $s_{p,i}$ to $s_{p,j}$ is denoted as:

$$t: s_{p,i} \rightarrow s_{p,j}, i \neq j, s_{p,i} \in S_p \text{ and } s_{p,j} \in S_p \cup \{\text{"unsecured"}\}$$

The *unsecured* state included in the target state set represents the unknown operational state that may be involved in the security evaluation applied to p

(see Section 4.2.3). To calculate the value of the impact of vulnerability exploitation, we need to assign numbers to each of the operation states. The value assignment of the states reflects the order of security impact: the bigger the value, the more state transition's impact. For example, the behavior $\langle authentication_check \rangle$ includes states such as $\{none = 0.1, bypass = 0.9\}$ which indicate no authentication will be checked (lowest secure level) and authentication check bypass (higher secure level) accordingly.

We denote $\ominus$ as the operator that determines the security difference (from now, we use “difference” to refer to the security difference) between two states. The impact of consequence of activity t can be defined as the difference between initial state $s_{p,i}$ and result state $s_{p,j}$:

$$imp_p = |s_{p,i} \ominus s_{p,j}|, \quad i \neq j$$

Assuming that the exploitation of vulnerability v will affect property p , its impact on p can be generally estimated as maximum difference between every available state of p :

$$imp_{p,t_v} = \max(|s_{p,i} \ominus s_{p,j}|), \quad i \neq j, s_{p,i} \in S_p \text{ and } s_{p,j} \in S_p \cup \{\text{"unsecured"}\}$$

Thus, the impact of v on application A will be estimated as:

$$\begin{aligned} imp_{A,t_v} &= \sum_{p_k \in G_v} w_{p_k} \cdot imp_{p_k,t_v} \\ &= \sum_{p_k \in G_v} w_{p_k} \cdot \max(|s_{p_k,i} \ominus s_{p_k,j}|) \end{aligned} \tag{1}$$

where w_{p_k} denotes the weight of connection between p_k and its adjacent node

in VPRG. Equation (1) also reflects the fact that the vulnerability will propagate its impact to every related property. Moreover, to distinguish the impact level of each property based on the relationship type to the vulnerable property, we define impact rate α for *cause*- and *consequence*- relations:

$$\alpha_{p_k} = \begin{cases} -\gamma & \text{if } p_k \text{ is active property in cause - relation} \\ +\eta & \text{if } p_k \text{ is passive property in consequence - relation} \end{cases}$$

Equation (1) will be rewritten as follows:

$$\begin{aligned} imp_{A,t_v} &= \sum_{p_k \in G_v} \alpha_{p_k} \cdot w_{p_k} \cdot imp_{p_k,t_v} \\ &= \sum_{p_k \in G_v} \alpha_{p_k} \cdot w_{p_k} \cdot \max(|s_{p_k.i} \ominus s_{p_k.j}|) \end{aligned} \quad (2)$$

4.3.3.2. Impact Calculation

Considering the cause-property and consequence-property analysis, the calculation of vulnerability impact (severity score) will be modified as follows.

We write G_v as VPRG of vulnerability v , G' to denote the cause subgraph and G'' to denote the consequence subgraph. To compute the severity score of vulnerability we need the following assumptions:

- The value of secure states of a property in common range of $[min, max] = [0,1]$ but the actual value will be decided by user in certain practical situation. E.g., a user can use secure states with $\{none = 0.1, bypass = 0.9\}$ for the behavior $\langle authentication_check \rangle$. In other case, this behavioral property can be assigned to have four possible states and values: $\{none = 0.1, guess = 0.15, user = 0.6, admin = 0.95\}$.

- The weight values w of connections between properties is given by $0 < w \leq 1$ which indicates how a property affects its consequence property.
- Impact rate α : $0 < |\alpha| \leq 1$. The actual value of α will vary depending on how the property relates to the other properties.

The consequence score will be calculated as following:

$$\begin{aligned}
 conseq &= \sum_{p_k \in G''} \alpha_{p_k} \cdot w_{p_k} \cdot card(p_k) \cdot imp_{p_k, t_v} \\
 &= (+\eta) \sum_{p_k \in G''} w_{p_k} \cdot card(p_k) \cdot imp_{p_k, t_v}
 \end{aligned} \tag{3}$$

For the cause subgraph, three factors need to be calculated separately. Firstly, the impact of disjunction of cause-properties is estimated as the average of disjunctive components p_i :

$$imp_{disj} = \frac{\sum_{p_i} w_{p_i} \cdot imp_{p_i, t_v}}{||p_i||} \tag{4}$$

Secondly, the impact of conjunction of cause-properties is estimated as:

$$imp_{conj} = \prod_{p_i} w_{p_i} \cdot imp_{p_i, t_v} \tag{5}$$

Thirdly, if cause-property is in a chain, the impact of its precedent properties will accumulate and the chain impact will affect the property at the last position in the chain. Therefore, the impact of cause-properties p_j in a chain is estimated as:

$$imp_{p_j} = w_{p_i} \cdot imp_{p_i, t_v} + imp_{p_j, t_v} \quad (6)$$

where p_i is precedent and adjacent node to p_j and the weight:

$$w_{p_i} = \begin{cases} w_{ij} & \text{if } p_i \rightarrow p_j \text{ and } p_i \text{ is not in combination} \\ 1 & \text{if } p_i \rightarrow p_j \text{ and } p_i \text{ is in combination } (\wedge, \vee) \end{cases} \quad (7)$$

If $w_{p_i} = 1$, which means p_i is in combination relation with other cause-properties precedent to p_j , the impact value imp_{p_i, t_v} of p_i is computed using either (4) or (5).

The cause impact of vulnerability will be computed as follows:

$$\begin{aligned} cause &= \sum_{p_k \in G'} \alpha_{p_k} \cdot imp_{p_k, t_v} \\ &= (-\gamma) \sum_{p_k \in G'} imp_{p_k, t_v} \end{aligned} \quad (8)$$

where the impact of direct cause-property imp_{p_k, t_v} is computed using (6). From (3) and (8), we compute the impact score of vulnerability v as follows:

$$impact = cause + conseq \quad (9)$$

In the following chapter, we apply the model to specified Web-based vulnerabilities. The proposed model is used to represent the vulnerability description and evaluate the severity of vulnerability.

CHAPTER 5

EXPERIMENTS

In the previous chapter, we presented a property-based vulnerability analysis model in (chapter 4, sections 1 and 2). A prototype of impact analysis and calculation based on the proposed model is presented in section 3. This chapter presents the application of the model with vulnerability data and experimental Web-based vulnerability scanner outputs.

The vulnerability data include SQL Injection vulnerability and Cross Site Scripting (XSS) vulnerability which are two of the most common Web-based vulnerabilities found recently. Cross Site Request Forgery (CSRF) is an example of vulnerability that exploits the logical weakness in human-application interactions. We also considered Apache Chunked-encoding Memory Corruption Vulnerability of Web server infrastructure system. Finally, we analyzed vulnerability scanner output and checked if the model is adaptable with scanner output.

The information of vulnerabilities is selected from different sources. Initially, we started the analysis with generic conceptual data from CWE. The specific data in CVE were used to depict the actual vulnerability. CVE presents analysis with real vulnerability exposure. Besides, an analysis article from SeachSecurity written by individual expert to evaluate XSS vulnerability was also taken into experiment. We will compare the model's descriptive capability with selected information sources at different levels of data abstraction and detail. Then, we calculate the impact score of the selected vulnerability.

5.1. Vulnerability Property and Property Relationship

Database Preparation

The database is a knowledge base storing the property sets and property relationship sets of every vulnerability. From collected information, we extract Web application properties which refer to the vulnerability. The sources also provide information from which we establish the application-based structure and determine the relationship among properties. We determined vulnerable property of the vulnerability. The VPRG graph and its Vulnerable Property Relation Matrix (VPRM) [82] representing the vulnerability will be created from properties and property relationship data.

In our experiments, we assumed that the related properties of vulnerability are manually analyzed. The conceptual meaning of description is also an issue when we analyse different sources. We need to manually verify the concept to ensure unified and unambiguous meaning before storing it to dataset. However, we decided to keep the concept description with least change and as close to the original descriptive information as possible. For example, “files with user-controllable inputs” and “Web pages with user-controllable inputs” refer to the same object, a file which is a Web page.

5.2. Experiments

CWE provides information which is commonly used among experts and end users, hence, it describes vulnerability in generic and standardized definitions. The selected vulnerabilities are described with VPRG and VPRM in Fig. 8 (SQL Injection vulnerability), Fig. 9 (XSS vulnerability), Fig. 10 (CSRF

vulnerability) and Fig. 11 (Apache Chunked-Encoding Memory Corruption Vulnerability) (E, B, PC and C are abbreviations for Entity, Behavior, Pre-condition and Communication, respectively).

While CWE only provides examples to let users understand the related concepts of vulnerability, expert can give more thorough analysis based on information from scanned results of the real scenario. End users usually do not or are unable to measure the severity of vulnerabilities and evaluate the related risk they might encounter when vulnerability is available in a Web application they are using. Hence, they should rely on the expert suggestion and the evaluation values provided by standard severity scoring techniques to assess the quality of the given application.

Figure 8. VPRM and VPRG of SQL Injection vulnerability: (a) CWE-89 and (b) described in SearchSecurity.com.

Figure 9. VPRM and VPRG of Cross Site Scripting (CWE-79): (a) simple version and (b) Extended version with variants on condition properties.

Figure 10. VPRM and VPRG of Cross Site Request Forgery (CWE-352)

We also need to specify secure states and cardinality of each property involving in vulnerability description. In our experiments, we accept a common assumption that when two values of extreme consequence states $\{min, max\} = \{0,1\}$ are applied for every property, then these properties have the impact of the same value: $imp = \max(|1 - 0|) = 1$. We also assumed that connection

A	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
(B) Handle chunked-encoding request_Apache WS (1)	0	0	0	0	1	1	0	0	0	0	0
(B) Authentication Apache WS (2)	1	0	0	0	0	0	0	0	0	0	0
(C) Network (3)	0	1	0	0	0	0	0	0	0	0	0
(B) Chunked encoding Access request (4)	0	0	1	0	0	0	0	0	0	0	0
(B) Provide service (5)	0	0	0	0	0	0	0	0	0	0	0
(B) execute code (6)	0	0	0	0	0	0	1	1	1	0	0
(E) Website (7)	0	0	0	0	0	0	0	0	0	0	0
(E) local user information (8)	0	0	0	0	0	0	0	0	0	0	0
(E) configuration information (9)	0	0	0	0	0	0	0	0	0	1	1
(C) Connection (10)	0	0	0	0	0	0	0	0	0	0	0
(E) back-end database (11)	0	0	0	0	0	0	0	0	0	0	0

Figure 11. VPRM and VPRG Apache Chunked-Encoding Memory Corruption Vulnerability (CVE-2002-0392)

weights are constant at 1 and the impact rate $\gamma = 0.3$ and $\eta = 0.5$, respectively.

For consequence properties, we assume that the cardinality is 1, which is acceptable by individual end user. Equations (3) to (9) will be used to calculate the impact score for vulnerabilities.

The severity scores of SQL Injection (Fig. 8a), XSS vulnerability (Fig. 9a and Fig. 9b), CSRF (Fig. 10) are presented in Table 6. The scores in Table 6 do not reflect the fact that many experts evaluate CRSF more severely than XSS. In this case, the description considered base evaluation framework, thus the score will illustrate the basic framework of vulnerability. In practical scenarios, the evaluation can be extended with additional properties according to detailed empirical analyses.

TABLE 6. SEVERITY SCORE OF SQL INJECTION, XSS AND CSRF VULNERABILITY (CWE)

Vulnerability	Severity score	Cause	Consequence
SQL Injection - CWE (Fig. 6a) - SearchSecurity (Fig. 6b)	1.7 (3.9)	0.3 (0.6)	2 (4.5)
XSS - simple (Fig. 7a) - extended (Fig. 7b)	1.7 (1.4)	0.3 (0.6)	2 (2)
CSRF	0.82	0.18	1
Apache Chunked-Encoding Memory Corruption Vulnerability	2.6	0.9	3.5

Detail analysis can provide property secure states and cardinality. From these, the end user may select secure state terms and assign values to state and determine the property secure state in actual situations. However, the secure state terms and values depend on specific skill and knowledge and actual scenario. In the experiment with Apache Chunked-Encoding Memory Corruption Vulnerability, we determined property secure states according to information from CVE database. The terms of each secure state, such as (inaccessible, accessible) for the entity “back-end database” were selected based on the related description of vulnerability consequence. In certain scenario, the pair (term, value) of secure state may be assigned differently by other users. Thus, it would be more relevant to measure the severity according to actual condition of property. Table 7 illustrates the properties secure states and instance cardinality of Apache Chunked-Encoding Memory Corruption Vulnerability (CVE-2002-0392). In this experiment, the vulnerability severity score is 2.6 (conseq=3.5, cause=0.9) and Fig. 11 shows its VPRG and VPRM.

The vulnerability scanners give output in different formats. We had scanned two sample Websites (demo.testfire.net and zero.webappsecurity.com) using

Acunetix, Cenzic HailStorm, HP WebInspect, IBM AppScan and N-Stalker and concentrated on XSS vulnerability. All above scanners generate output focusing on the location of vulnerability (path and name of vulnerable file), request with sample malicious parameters sent to that location and response with detected sign of XSS vulnerability which are adaptable with properties (1), (2), (3), (4), and (5) in Fig. 9.

TABLE 7. SECURE STATE OF CVE-2002-0392 VULNERABILITY PROPERTIES

Property	cardinal	Secure state
(BH) Handle chunked-encoding request _Apache WS (1)	(0..*)	<ul style="list-style-type: none"> - secured (<i>direct solution is officially available</i>) - partial secured (<i>direct solution is unofficially available or under testing</i>) - not secured (<i>no solution confirmed</i>)
(BH) Authentication _Apache WS (2)	(0..*)	<ul style="list-style-type: none"> - advanced authentication (<i>high complexity</i>) - basic authentication (<i>medium complexity</i>) - none
(C) Network (3) (data transfer via at least 1 network)	(1..n)	<ul style="list-style-type: none"> - secured - unsecured
(BH) Chunked encoding Access request (4)	(1..n)	<ul style="list-style-type: none"> - secured (<i>valid</i>) - unsecured (<i>security invalid</i>)
(BH) Provide service (5) (at least 1 service session will be provided)	(1..n)	<ul style="list-style-type: none"> - low latency - medium latency - high latency - stop
(BH) execute code (6)	(1..*)	<ul style="list-style-type: none"> - non-executable - high-restricted execution (<i>guest permission</i>) - restricted execution (<i>normal user</i>) - execution (<i>admin privileges</i>)
(E) Website (7)	(1..n)	<ul style="list-style-type: none"> - content-safe - content-alternation
(E) local user information (8)	(0..*)	<ul style="list-style-type: none"> - hidden - revealed
(E) configuration information (9)	(0..*)	<ul style="list-style-type: none"> - hidden - revealed
(C) Connection (10)	(0..*)	<ul style="list-style-type: none"> - protected - unprotected
(E) back-end database (11)	(0..n)	<ul style="list-style-type: none"> - inaccessible - accessible

CHAPTER 6

CONCLUSION

6.1. Summary

This project has established a property-based description model that supports analyzing and describing vulnerabilities of Web-based applications. The combination of information from several sources and scanners output may give a coverage description of vulnerability in both abstract and detail level. However, the diversity of vulnerability detection algorithms and methods makes it difficult to establish a useful coverage solution, particularly if there is no expert. The model developed in this project is based on the application properties and their relationship, and it is proposed to work with different sources of vulnerability information and provides a unified description that can be used among users. The graph-theoretic model allows the user to present the nature of vulnerability and determine the cause and consequence of vulnerability. By analyzing the relationship of properties in the cause and consequence subgraph of VPRG, the graph model can be used to evaluate the impact and to calculate impact score of vulnerability.

The project framework for Web-based vulnerability analysis has been made for scalability and efficiency.

This work has been submitted to ACM Transactions on Information and System Security (TISSEC).

6.2. Future Work

The Web-based vulnerability analysis considers the description of vulnerability and its variants. We proposed that certain combinations of application-based properties may present the variants of vulnerability and its exploitation. Another problem is the relationship with other vulnerabilities in order to establish a complete evaluation of vulnerability. It is also expected that the estimation capability which can foresee the possible vulnerability based on existing information would be integrated in the analysis system. Finally, the knowledge obtained from vulnerability data sources using the model must be used to analyze scanner results for a comprehensive report and support diagnosis.

In our recent work, we are using Graph Similarity Algorithm to recognize vulnerability property matching in VPRG Model. We reviewed the object-to-object similarity measurements and graph matching problems. We focused on the Similarity Flooding algorithm [87] and proposed a matching scheme that is relevant to our VPRG model. We discussed and performed experiments to show that the graph matching with SF-based scheme can be used to analyze the relationship between vulnerability description at both conceptual level (CWE) and detailed level (CVE). The relationship between two selected vulnerabilities (CSRF and XSS) can also be seen from the output of graph matching. The results of this work will be considered as a potential approach to establish the taxonomy of vulnerability based on VPRG model in our future research. This work has been submitted to the CACS2010 conference [88].

In the next stage, we will concentrate on the following problems:

- Continue investigating the capability of the proposed property-based VPRG model in expressing the relations between vulnerabilities based on the graph similarity approach.
- Extend the relationship analysis to investigate the capability of the property-based VPRG model in describing vulnerability variants.
- Using the property-based VPRG model to obtain the knowledge base of vulnerability analysis and support complete and detailed description of Web vulnerabilities based on scanning result and existing vulnerability databases.
- Test the application of the model in vulnerability impact measurement.

The improvement and extension of the model may be needed to make the model applicable to the problems stated above.

BIBLIOGRAPHY AND REFERENCES

- [1] J. Grossman, "WhiteHat Website Security Statistics Report," WhiteHat Security October 2007.
- [2] J. Grossman, "WhiteHat Website Security Statistics Report," WhiteHat Security August 2008.
- [3] IBM, "Cyber Attacks On The Rise: IBM 2007 Midyear Report," IBM Internet Security Systems™ X-Force® Research and Development 2007.
- [4] K. Raina. *Trends in Web Application Security* [Online]. Available: <http://www.securityfocus.com/print/infocus/1809>. Last accessed: 30 October 2007.
- [5] nCircle, "nCircle Vulnerability Scoring System," nCircle Network Security, White Paper 2006.
- [6] Adobe. *Severity Rating System for Security Bulletins*. Available: http://www.adobe.com/devnet/security/security_zone/severity_ratings.html. Last accessed: December 2010.
- [7] Cenizic. *Cenizic Product Guide - Cenizic Hailstorm Enterprise ARC-HARM Score*. Available: <http://doc.cenizic.com/sadoc9x14ba847/harm.htm>. Last accessed: 28 September 2009.
- [8] J. R. Jones, "Estimate Software Vulnerabilities," *IEEE Security & Privacy Magazine*, vol. 5, 2007, pp. 28-32, doi: 10.1109/MSP.2007.81
- [9] P. Mell, K. Scarfone, and S. Romanosky. (2007). A Complete Guide to the Common Vulnerability Scoring System Version 2.0. doi: N/A. Available: <http://www.first.org/cvss/cvss-guide.html>
- [10] Microsoft. *Microsoft Security Response Center Security Bulletin Severity Rating System*. Available: <http://www.microsoft.com/technet/security/bulletin/rating.mspx>. Last accessed: December 2010.
- [11] Secunia. *Secunia Advisories - Vulnerability Intelligent*. Available: <http://secunia.com/advisories/terminology/> Last accessed: 19 August 2010.
- [12] US-CERT. *Vulnerability Notes Database Field Descriptions*. Available: <http://www.kb.cert.org/vuls/html/fieldhelp#metric>. Last accessed: December 2010.
- [13] Insecure. *Top 10 Web Vulnerability Scanners* [Online]. Available: <http://sectools.org/web-scanners.html>. Last accessed: 30 August 2007.
- [14] WASC. *Web Application Security Scanner List*. Available: <http://projects.webappsec.org/Web-Application-Security-Scanner-List>. Last accessed: 30 November 2009.
- [15] NIST. *Web Application Vulnerability Scanners - SAMATE*. Available: http://samate.nist.gov/index.php/Web_Application_Vulnerability_Scanners.html. Last accessed: December 2010.
- [16] NIST. *National Vulnerability Database Home*. Available: <http://nvd.nist.gov>. Last accessed: December 2010.
- [17] CVE. *CVE - Common Vulnerabilities and Exposures (CVE)* [Online]. Available: <http://cve.mitre.org/>. Last accessed: December 2010.
- [18] Finjan. *Web Security Threats and Latest Web Vulnerabilities*. Available: <http://www.finjan.com/Content.aspx?id=566>. Last accessed: December 2010. (redirect to <http://www.m86security.com/labs/latest-web-vulnerabilities.asp>).
- [19] Secunia. *Vulnerability and Virus Information - Secunia* [Online]. Available: <http://secunia.com/>. Last accessed: 6 January 2009.
- [20] SecurityFocus. *Bugtraq Mailing list* [Online]. Available: <http://www.securityfocus.com/archive/1>. Last accessed: December 2010.

- [21] SecurityFocus. *Vulnerabilities list* [Online]. Available: <http://www.securityfocus.com/vulnerabilities>. Last accessed: December 2010.
- [22] US-CERT. *Vulnerability Notes Database*. Available: <http://www.kb.cert.org/vuls>. Last accessed: December 2010.
- [23] G. C. Hunt, J. R. Larus, D. Tarditi, and T. Wobber, "Broad New OS Research: Challenges and Opportunities," in the *HotOS X, Tenth Workshop on Hot Topics in Operating Systems*, Santa Fe, NM, 2005, p. 15, doi: 10.1.1.118.1365.
- [24] R. P. Abbott, J. S. Chin, J. E. Donnelley, W. L. Konigsford, S. Tokubo, and D. A. Webb, "Security Analysis and Enhancements of Computer Operating Systems," Lawrence Livermore Laboratory, Technical Report TR NBSIR-76-1041, April 1976.
- [25] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A Taxonomy of Computer Program Security Flaws, with Examples," Center for Computer High Assurance Systems Information Technology Division, Technical Report NRL/FR/5542--93-9591, November 1993.
- [26] M. Bishop, "A Taxonomy of UNIX System and Network Vulnerabilities," Department of Computer Science University of California at Davis, Technical Report SCE-95-10, May 1995.
- [27] M. Bishop, "Vulnerabilities Analysis," in the *2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99)*, West Lafayette, Indiana, USA, 1999, pp. 125-136, doi: 10.1.1.39.6062.
- [28] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," presented at the 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, 2007.
- [29] J. Steffan and M. Schumacher, "Collaborative Attack Modeling," in the *2002 ACM Symposium on Applied Computing (SAC 2002)*, Madrid, Spain, 2002, pp. 253-259, doi: 10.1145/508791.508843.
- [30] C. V. Berghe, J. Riordan, and F. Piessens, "A Vulnerability Taxonomy Methodology applied to Web Services," in the *10th Nordic Workshop on Secure IT-systems (NORDSEC 2005)*, Tartu, Estonia, 2005, doi: 10.1.1.61.286.
- [31] A. Bazaz and J. D. Arthur, "Towards A Taxonomy of Vulnerabilities," in the *40th Annual Hawaii International Conference on System Sciences (HICSS-40 2007)*, Waikoloa, HI, USA, 2007, pp. 163a - 163a, doi: 10.1109/HICSS.2007.566
- [32] A. Bazaz, J. D. Arthur, and J. G. Tront, "Modeling Security Vulnerabilities: A Constraints and Assumptions Perspective," in the *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC'06)*, Indianapolis, IN, USA 2006, pp. 95-102, doi: 10.1109/DASC.2006.35
- [33] M. D. Aime, A. Atzeni, and P. C. Pomi, "The Risks With Security Metrics," presented at the 4th ACM Workshop on Quality of Protection, Alexandria, Virginia, USA, 2008.
- [34] Å. J. Holmgren, "Using Graph Models to Analyze the Vulnerability of Electric Power Networks," *Risk Analysis*, Vol. 26, No. 4, pp. 955-969, August, 2006, doi: 10.1111/j.1539-6924.2006.00791.x.
- [35] M. E. Rafiei, R. Jalili, and H. Mousavi, "Vulnerability Analysis through a Graph-based Protection System," *IJCSNS International Journal of Computer Science and Network Security*, vol. 6, December 2006, pp. 311-319, doi: N/A.
- [36] L. P. Swiler, C. Phillips, and T. Gaylor, "A Graph-Based Network-Vulnerability Analysis System," Sandia National Labs., Albuquerque, NM (United States), Technical Report 01 January 1998.
- [37] L. Wang, A. Singhal, and S. Jajodia, "Toward Measuring Network Security Using Attack Graphs," presented at the ACM Workshop on Quality of Protection, Alexandria, Virginia, USA, 2007.

- [38] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," presented at the 1st Conference on India Software Engineering, Hyderabad, India, 2008.
- [39] S.-W. Woo, O. H. Alhazmi, and Y. K. Malaiya, "An Analysis Of The Vulnerability Discovery Process In Web Browsers," in the *10th IASTED International Conference on Software Engineering and Applications*, Dallas, TX, USA, 2006, pp. 172-177, doi: 10.1.1.61.8329.
- [40] W. G. J. Halfond, A. Orso, and P. Manolios, "Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks," in the *14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ACM SIGSOFT 2006/FSE-14)*, Portland, Oregon, USA, 2006, pp. 175-185, doi: 10.1145/1181775.1181797.
- [41] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing Web Application Code by Static Analysis And Runtime Protection," in the *13th International Conference on World Wide Web*, New York, NY, USA, 2004, pp. 40-52, doi: 10.1145/988672.988679.
- [42] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans, "Automatically Hardening Web Applications Using Precise Tainting," in the *20th IFIP International Information Security Conference*, Makuhari-Messe, Chiba, Japan, 2005, doi: 10.1.1.65.2467, 10.1.1.58.8282.
- [43] A. K. Ghosh, T. O'Connor, and G. McGraw, "An Automated Approach for Identifying Potential Vulnerabilities in Software," in the *1998 IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1998, pp. 104-114, doi: 10.1109/SECPRI.1998.674827
- [44] D. Hurst. *Asking the Right Question: Penetration Testing vs. Vulnerability Analysis Tools, Which Is Best?* [Online]. Available: <http://www.infosecwriters.com/texts.php?op=display&id=537>. Last accessed: 31 October 2007.
- [45] M. Cova, V. Felmetsger, and G. Vigna, "Vulnerability Analysis of Web-based Applications," in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds., ed: Springer Berlin Heidelberg, 2007, pp. 363-394.
- [46] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A Taxonomy of Computer Program Security Flaws, with Examples," Center for Computer High Assurance Systems Information Technology Division, Technical Report NRL/FR/5542--93-9591, 1994.
- [47] S. Christey and C. Harris. *Introduction to Vulnerability Theory (Version 1.0.1 ed.)* [draft]. Available: http://cwe.mitre.org/documents/vulnerability_theory/intro.html. Last accessed: December 2010.
- [48] S. Shah. *Vulnerability Scanning Web 2.0 Client-Side Components* [Web page]. Available: <http://www.symantec.com/connect/articles/vulnerability-scanning-web-20-client-side-components>. Last accessed: 18 March 2010.
- [49] SecureEnterprise2.0Forum, "Top Web 2.0 Security Threats," Security Enterprise 2.0 Forum, Industry Report 17 February 2009.
- [50] A. A. Al-Tameem, "The Impact of AJAX Vulnerability in Web 2.0 Applications," *Journal of Information Assurance and Security*, vol. 3, 2008, pp. 240-244, doi: N/A.
- [51] B. Chess, Y. T. O'Neil, and J. West, "JavaScript Hijacking," Fortify Software, White paper 12 March 2007.
- [52] B. Hoffman, "Ajax (in)security," presented at the Black Hat USA, 2006.
- [53] Acunetix. *Scan AJAX and Web 2.0 Technologies for Vulnerabilities*. Available: <http://www.acunetix.com/vulnerability-scanner/ajax.htm>. Last accessed: 16 March 2010.

- [54] M. Dowd, J. McDonald, and J. Schuh, in *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities.*, 1 ed: Addison Wesley Professional, 2006.
- [55] M. Stamp, *Information Security: Principles and Practice*: John Wiley & Sons, 2006.
- [56] N. Jovanovic, C. Kruegel, and E. Kirda, "Precise Alias Analysis for Static Detection of Web Application Vulnerabilities," in the *2006 Workshop on Programming Languages and Analysis for Security (PLAS '06)*, Ottawa, Ontario, Canada, 2006, pp. 27-36, doi: 10.1145/1134744.1134751.
- [57] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short paper)," in the *2006 IEEE Symposium on Security and Privacy (S&P'06)*, Berkeley/Oakland, California, USA, 2006, pp. 258-263, doi: 10.1109/SP.2006.29.
- [58] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "SecuBat: A Web Vulnerability Scanner," in the *15th International Conference on World Wide Web (WWW 2006)*, Edinburgh, Scotland, 2006, pp. 247 - 256, doi: 10.1145/1135777.1135817.
- [59] O. H. Alhazmi and Y. K. Malaiya, "Quantitative Vulnerability Assessment of System Software," in the *Annual Reliability and Maintainability Symposium*, 2005, pp. 615-620, doi:
- [60] H. T. Le and P. K. K. Loh, "Evaluating AVDL Descriptions for Web Application Vulnerability Analysis," in the *IEEE International Conference on Intelligence and Security Informatics 2008 (ISI 2008)*, Taipei, Taiwan, 2008, pp. 279-281, doi: 10.1109/ISI.2008.4565084.
- [61] H.-T. Le and P. K. K. Loh, "Realizing Web Application Vulnerability Analysis via AVDL," in the *10th International Conference on Enterprise Information Systems (ICEIS 2008)*, Barcelona, Spain, 2008, pp. 259-265, doi: N/A.
- [62] OASIS. *Application Vulnerability Description Language v1.0*. Available: <http://www.oasis-open.org/committees/download.php/7145/AVDL%20Specification%20V1.pdf>. Last accessed: December 2010.
- [63] OASIS. *Application Security Standards*. Available: <http://xml.coverpages.org/appSecurity.html>. Last accessed: December 2010.
- [64] T. Duong and J. Rizzo, "Flickr's API Signature Forgery Vulnerability," Advisory 28 September 2009.
- [65] J. Topf. *The HTML Form Protocol Attack*. Available: <http://www.remote.org/jochen/sec/hfpa/hfpa.pdf>. Last accessed: July 2010.
- [66] CVE. *CVE - Common Vulnerabilities and Exposures (CVE)* [Online]. Available: <http://cve.mitre.org/>. Last accessed: December 2010.
- [67] ha.ckers.org. *XSS (Cross Site Scripting) Cheat Sheet*. Available: <http://ha.ckers.org/xss.html>. Last accessed: December 2010.
- [68] NIST. *National Vulnerability Database Home*. Available: <http://nvd.nist.gov>. Last accessed: December 2010.
- [69] SANS. *SANS Internet Storm Center; Cooperative Network Security Community - Internet Security - isc* [Online]. Available: <http://isc.sans.org/>. Last accessed: December 2010.
- [70] US-CERT. *Vulnerability Notes Database*. Available: <http://www.kb.cert.org/vuls>. Last accessed: December 2010.
- [71] ha.ckers.org. *ha.ckers.org Web Application Security Lab*. Available: <http://ha.ckers.org/>. Last accessed: December 2010.
- [72] MilwOrm. *milw0rm - exploits : vulnerabilities : videos : papers : shellcode* [Online]. Last accessed: 2007. (MilwOrm <http://www.milw0rm.com/> was announced to closed on 08 July 2009).

- [73] OffensiveComputing. *Offensive Computing | Community Malicious code research and analysis* [Online]. Available: <http://www.offensivecomputing.net/>. Last accessed: December 2010.
- [74] PacketStorm. *./[packet storm]:*. [Online]. Available: <http://packetstormsecurity.org/>. Last accessed: December 2010.
- [75] W. G. J. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," in the *IEEE International Symposium on Secure Software Engineering (ISSSE 2006)* Arlington, VA, USA, 2006, doi: 10.1.1.95.2968.
- [76] CenZic. *CenZic Product Guide - CenZic Hailstorm Enterprise ARC-HARM Score*. Available: <http://doc.cenzic.com/sadoc9x14ba847/harm.htm>. Last accessed: 28 September 2009.
- [77] H. T. Le and P. K. K. Loh, "Unified Approach to Vulnerability Analysis of Web Applications," in the *AIP Conference Proceedings, International Electronic Conference on Computer Science*, 2008, pp. 155-159, doi: 10.1063/1.3037041.
- [78] D. Subramanian, H. T. Le, and P. K. K. Loh, "Fuzzy Heuristic Design for Diagnosis of Web-Based Vulnerabilities," in the *2009 Fourth International Conference on Internet Monitoring and Protection (ICIMP'09)*, Venice/Mestre, Italy, 2009, pp. 103-108, doi: 10.1109/ICIMP.2009.25
- [79] D. Subramanian, H. T. Le, and P. K. K. Loh, "Assuring Quality in Vulnerability Reports for Security Risk Analysis," *International Journal on Advances in Security*, vol. 2, 2009, pp. 226-241, doi: N/A.
- [80] D. Subramanian, H. T. Le, P. K. K. Loh, and A. B. Premkumar, "An Empirical Vulnerability Remediation Model," in the *IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS2010)*, Beijing, China, 2010, pp. 376-380, doi: 10.1109/WCINS.2010.5541803.
- [81] D. Subramanian, H. T. Le, P. K. K. Loh, and A. B. Premkumar, "Quantitative Evaluation of Related Web-based Vulnerabilities," in the *4th IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2010)*, Singapore, 2010, pp. 118-125, doi: 10.1109/SSIRI-C.2010.30
- [82] H. T. Le, D. Subramanian, W. J. Hsu, and P. K. K. Loh, "An Empirical Property-Based Model for Vulnerability Analysis and Evaluation," in the *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific. Workshop of Secure Service Computing 2009 (SSC'09)*, Singapore, 2009, pp. 40-45, doi: 10.1109/APSCC.2009.5394144.
- [83] H. T. Le, D. Subramanian, W. J. Hsu, and P. K. K. Loh, "Scoring Web-based Vulnerability Impact using Property-Based Vulnerability Model," in the *6th International Symposium on Web and Mobile Information Services (WAMIS), In conjunction with the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications (AINA2010)*, Perth, Australia, 2010, pp. 431-436, doi: 10.1109/WAINA.2010.123.
- [84] US-CERT. *Vulnerability Note VU#476267*. Available: <http://www.kb.cert.org/vuls/id/476267>. Last accessed: July 13 2009.
- [85] W. Alcorn. (2006, July 15, 2009). Inter-Protocol Communication. doi: N/A. Available: <http://www.ngsconsulting.com/research/papers/InterProtocolCommunication.pdf>
- [86] S. Palmer, *Web Application Vulnerabilities: Detect, Exploit, Prevent*. Syngress, 2007.

- [87] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm (Extended Technical Report)," Stanford InfoLab, Technical Report 2001.
- [88] H.-T. Le, P. K. K. Loh, and W. J. Hsu, "Using Graph Similarity Algorithm To Recognize Vulnerability Property Matching in VPRG Model," presented at the 2010 International Congress on Computer Applications and Computational Science (CACCS2010), Singapore, 2010.