A UNIFIED CYBER-ENHANCED APPROACH FOR DETECTING CROSS-SITE SCRIPTING ATTACKS ON WEB APPLICATIONS

BHANUKIRAN GURIJALA

Doctoral Program in Computer Science

APPROVED:

Ann Q. Gates, Ph.D., Chair

Salamah I. Salamah, Ph.D., Chair

Luc Longpré, Ph.D.

Natalia Villanueva-Rosales, Ph.D.

Miguel Velez-Reyes, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

DEDICATION

To My Beloved Family,
Especially To
Mother, Lavanya Gurijala and
Father, Narasimhulu Venkata Gurijala

A UNIFIED CYBER-ENHANCED APPROACH FOR DETECTING CROSS-SITE SCRIPTING ATTACKS ON WEB APPLICATIONS

by

BHANUKIRAN GURIJALA, M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

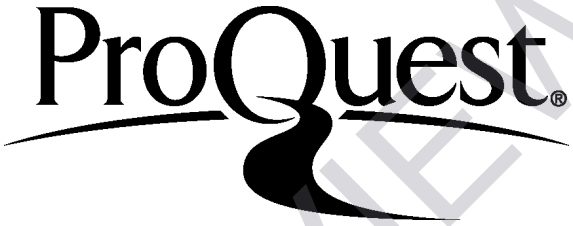
for the Degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

August 2016



ACKNOWLEDGEMENTS

I would like to express my utmost gratitude to my advisor and mentor Dr. Ann Q. Gates. Her professionalism and meticulousness in pursuit of excellence are a great source of inspiration that I continue to learn and pass forward to others in my career. I would like to express my greatest gratitude to my co-advisor and mentor Dr. Salamah I. Salamah.

I am very grateful to my committee for their invaluable contributions, feedback, and support: Dr. Luc Longpré, for his invaluable feedback about security domain and automata; Dr. Natalia Villanueva-Rosales, for her invaluable feedback in knowledge representation; Dr. Miguel Velez-Reyes, for his consistent support and invaluable feedback with encouraging words.

Additionally, I want to thank the professors and staff of the Computer Science Department and the CyberShARE Center of Excellence at the University of Texas at El Paso for their support and enhancing my learning experience.

I would also like to thank my father, mother, and sister for their love and support. My father, Narasimhulu Venkata Gurijala, has always been supportive of my decisions and plans. My mother, Lavanya Gurijala, has always encouraged me to chase my dreams and supported me in their pursuit. My sister, Indumathi Gurijala, has been supportive and encouraging. I would like to thank my best friend Akash Agarwal and his wife Deepavali Chakravarti who are no less than my family. They have continuously supported me and encouraged me through thick and thin.

This work is partly supported by CAHSI and CyberShare through grants from National Science Foundation (NSF) CNS-1042341 and HRD-1242122. Any opinions, findings, and conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the NSF.

ABSTRACT

Cyber-security is one of our nation's most critical security priorities, and its importance continues to grow with the pervasiveness of computers and Web-based applications. In particular, cross-site scripting (XSS) is one of the most common and dangerous types of injection attacks that exploit input validation vulnerabilities. XSS has intensified due to: 1) lack of extensive security domain knowledge of software engineers who are involved in building and/or maintaining Web-applications; and 2) lack of proper software development processes focused on security, resulting in fixes to security vulnerabilities late in the software development lifecycle. Indeed, the cost benefits of removing defects, in particular security-related faults, earlier in the lifecycle is well documented. The **research goal** is to reduce successful XSS attacks through a unified approach that identifies malicious and suspicious inputs/outputs based on customized application-specific knowledge. The Intrusion Detection Approach (IDA), which is defined in this dissertation, captures XSS-related domain knowledge from national catalogs of attack patterns and uses it to generate application-specific XSS patterns for monitoring IO by integrating technologies and techniques such as ontologies, provenance, formalizations and security-related domain knowledge. The work hosts a security knowledge base that is easily maintainable whenever new attacks or new ways of launching attacks come into use. Updating the security knowledge base results in monitoring, identifying, and preventing XSS attacks without any changes to the Web application. Risk analysis combined with provenance provides a unique way of prioritizing formalized patterns based on sensitivity level of assets and trends of threats. Using the XSSMon tool, which realizes the IDA, the author conducted a case study on two versions of a commercial Web application to compare the effectiveness of XSSMon. The results showed that XSSMon had higher success rates in identifying XSS-related attacks. Specifically, the overall success rates were 4.79% and 28.01% for the original and latest version of the Web application, respectively, and 88.36% and 100% for the initial and latest version of XSSMon, respectively. The results of this case study can be extended to other Web applications that accept similar equivalence classes of IO.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 PROBLEM STATEMENT AND GOAL

Cyber-security can be defined as security measures applied to computers to provide a desired level of protection with respect to confidentiality, integrity, and availability (**CIA**) concerns. Confidentiality refers to the property that data should only be viewable by authorized parties; integrity to the principle that only authorized users are allowed to change data; and availability to the principle that data and computer resources will always be available to authorized users [1]. Cyber attacks undermine the CIA concerns or information resident on it. The importance of cyber-security is grows as the integration of computers into more and more aspects of modern life continues. This in turn has increased cyber-attacks and the need to mitigate them, in particular in dynamic Web applications that accept input from the user and performs actions based on the input. The heart of the issue is the introduction of distrusted content into a dynamic page, where neither the Web site nor the client has enough information to detect the event and take protective actions. Such distrusted content can be introduced through malicious code provided by one client for another client, or through malicious code sent by a client for itself.

Distrusted content or malicious scripts result in one of the most common application-level attacks known as Cross Site Scripting (**XSS**). XSS was the second most prevalent consequence of vulnerability exploitation for the first half of 2013 at 18% [2] and was ranked third for the entire year of 2013 [3] [4] and has consistently stayed in the top three risks for Web applications since 2002 [2-6]. With XSS, every input and output has the potential to be an attack vector, which does not occur with other vulnerability types. In addition, XSS has numerous subtleties and variants. The **goal** of the work is to reduce successful XSS attacks through a unified approach that identifies malicious and suspicious inputs/outputs based on customized application-specific knowledge. This dissertation introduces the Intrusion Detection Approach (**IDA**), which realizes the goal by using cyber-enhanced technologies, i.e., ontologies and provenance, to manage application-specific knowledge for anomaly detection and XSS knowledge from national vulnerability data bases for

misuse detection. The customized and formalized knowledge can be used monitor inputs and outputs to Web applications to determine if they are capable of launching XSS attacks.

1.2 MOTIVATION

Cyber-attacks are placed among top five risks the world is likely to face over the next decade [7] [8]. Cyber-attacks, in particular XSS, pose a severe threat to Web-application security due to: a) lack of extensive security domain knowledge in software engineers who are involved in building and maintaining Web-applications; and b) lack of proper software development processes focused on security, resulting in fixes to security vulnerabilities late in the software development life cycle (SDLC). Many approaches have been presented for detecting and preventing security breaches; however, the approaches are reactive. Section 2 describe the approaches to detect security breaches.

By capturing, maintaining, and sharing XSS-related knowledge, it is posited that XSS attacks can be prevented by supporting application-specific customization of XSS knowledge to create an intermediate layer between Web applications and its users. Furthermore, this would address the aforementioned challenges by providing customization of existing knowledge about XSS attacks for a Web application leading to enhanced security. In addition, the approach will provide the ability to restructure and customize the knowledge to address changes to the Web application over the course of its life cycle by keeping abreast with the new and emerging weaknesses and attack patterns.

1.3 CHALLENGES

The challenges for developing a comprehensive security solution for XSS attack are as follows [9]:

1. **System requirements needed for XSS attack**
   - **i.** Malicious hackers have developed many tools that can launch XSS attacks by bypassing the safeguards employed by the Web applications. The entry points of

vulnerable XSS Web applications can be found using automated tools. PHP Charset Encoder is an example tool to launch attacks.

ii. The most basic data manipulations that exploit the vulnerability and launch XSS attacks are simple to perform. No special tools are needed to launch XSS attacks. Only a Web browser is needed.

iii. New evasive mechanisms can easily be found on the hacker's discussion forums, where new hacking attempts and success stories are discussed.

2. **Social factors**

i. XSS vulnerabilities arise due to coding practices. Web applications are developed by programmers with varied experience and with little or no knowledge of security needs of an application. Coding vulnerabilities vary from site to site, and there is no single patch available to fix all XSS vulnerabilities.

ii. Web applications are usually maintained by a team that is generally not the same team that developed the application, which increases the chances of introducing new XSS vulnerabilities with each change request.

3. **Diverse business needs**

i. Businesses prioritize increasing the customer base to increase the revenue. The advent of technologies, such as AJAX, Web Services and Web 2.0, have contributed to more aggressive schedules and a practice of prioritizing features that are not inclusive of security. Usually, requirements evolve and change during implementation that forces the developers to change functionality of the application without concern for changes to security mechanisms.

ii. Web applications are developed to meet diverse business needs. Therefore, the security mechanism applicable for one Web application may not be applicable for the other.

The approach aims to address the aforementioned challenges.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Definitions

Definitions	Meaning
Blacklist or Block-list	A list of discrete entities that have been previously determined to be associated with malicious activity
Fuzz testing or Fuzzing	Black-box software testing technique that involves using malformed or semi-malformed data injection in an automated fashion.
Fuzzer	Program that injects automatically the fuzz inputs into a program.
Pattern	The regular and repeated way in which something happens or is done
Payload	The malicious code that performs a destructive operation
Whitelist	A list of discrete entities that are known to be benign and are approved for use

Acronyms

Acronyms	Meaning
AOP	Aspect Oriented Programming
CAPEC	Common Attack Pattern Enumeration and Classification
CSRF	Cross Site Reference Forgery
CVE	Common Vulnerabilities and Exposures
CWE	Common Weaknesses Enumeration
DFA	Deterministic Finite Automata
DOM	Document Object Model
DSI	Document Structure Integrity
FPSIV	Five Primary Security Input Validation
FSA	Finite State Automata
FSM	Finite State Machines
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IRM	Inline Reference Monitor
IVV	Input Validation Vulnerability
MBT	Model Based Testing
MDD	Model Driven Development
MFE	Malicious File Execution
NFA	Non-deterministic Finite Automata
OWASP	Open Web Application Security Project
OWL	Web Ontology Language
PQL	Program Query Language
QAS	Quality Attribute Scenarios
RFI	Remote Malicious File Inclusion
SAW	Semantic Abstract Workflow
SDLC	Software Development Life Cycle
SOA	Service Oriented Architecture
SOP	Same Origin Policy
SQL	Structured Query Language

Acronyms	Meaning
SVM	Support Vector Machines
UIV	User Input Validation
WfMC	Workflow Management Coalition
WfMS	Workflow Management Systems
XCS	Cross Channel Scripting
XML	Extensible Markup Language
XSD	XML Schema Definition
XSS	Cross Site Scripting
ZAP	Zed Attack Proxy

Abbreviations

Abbreviations	Meaning
e.g.	For example
i.e.	That is
ID	Identification

1.4 ORGANIZATION

The organization of rest of the dissertation document is as follows: Chapter 2 presents the background information to enhance understanding of the rest of the document; it also includes a comprehensive survey of XSS detection and prevention approaches. Chapter 3 presents the approach; it details the steps of the approach needed to accomplish the research goal. Chapter 4 presents the design and implementation details of the XSSMon tool developed based on the described approach. Chapter 5 presents the setup, design, and results of the case study and their analysis that support evaluation of the XSSMon tool to evaluate tool designed based on the approach. Chapter 6 compares the more closely aligned systems described in Chapter 2 to the IDA approach. Chapter 7 presents the conclusion, which includes a summary, significance, limitations of the work and future directions of the effort. Appendix A includes a detailed description of the techniques and approaches for detecting and preventing XSS attacks surveyed. Appendix B presents the test suite for performing unit testing of the designed tool. Appendix C presents the CRC cards for the tool design. Appendix D presents the details for each of the methods including pre- and post-conditions and test design strategy for unit testing.

CHAPTER 2: BACKGROUND

This section provides background information to enhance understanding of the rest of the document. This section is organized as follows: Section 2.1 provides an overview of cyber-security and cyber-attacks from a historical perspective, including an overview of Web application vulnerabilities and the main categories of XSS; Section 2.2 provides an overview of techniques, methods, and tools that are available for detecting and/or preventing XSS vulnerabilities; Section 2.3 provides an overview of cyber-enhanced techniques and technologies, including an overview of ontologies and their current usage in capturing security related domain knowledge and an overview of attack-trees and their usage in different phases of SDLC with respect to software security.

2.1 HISTORICAL OVERVIEW OF CYBER-SECURITY AND CYBERATTACKS

Early computer systems offered high security with relatively less functionality and availability as compared to the current software systems [1]. As software vendors increased functionality, moved to PCs and later to distributed computing and Web services, data availability has increased by orders of magnitude, thereby marking the increase in issues of confidentiality and integrity. The driving principle behind software development has been functionality and not security. The basic design of the Internet was built around shared access and trust with security measures being an afterthought. Many protocols in wide use mostly rely on trust and offer little, if any, security to their users. This model made sense when the Internet was first developed and information being transferred was not critical. Today, the Internet is used to transfer information between people, their banks, their brokers, businesses and government entities mostly using Web applications, which makes this information susceptible to cyber-attacks, identity thefts and phishing attacks.

The earliest Web applications used the CGI protocol to interface with external applications. The applications shared security issues that resulted from the common text parsing approaches used. Applications were often vulnerable to input validation weaknesses, buffer overflows, and

denial of service attacks. As the Web gained popularity, Web browsers and HTTP protocol evolved. Commercial Web-server software emerged that allowed Web developers to produce custom plugins or extensions and filters that ran within the Web server process. This allowed developers to interact with a Web request at multiple stages in the serving process, which exposed Web servers to potentially vulnerable code and malicious user input, increasing the risk of Web server crashes and imported security vulnerabilities. Attacks against Web servers were generally concentrated on the core Web-server code and supporting libraries, or manipulated URL elements and injection of unexpected user data. The attacks usually fell into standard vulnerability categories such as: buffer overflows, sample code, input validation attacks, format string attacks, canonicalization attacks, encoding attacks, privilege escalation, form tampering, user created content, XSS, SQL injection, insecure direct object reference, remote malicious file inclusion (RFI), cross-site request forgery (CSRF), access control weaknesses, authentication and session management failures, data confidentiality failures, and poor error handling [10].

2.1.1 Overview of Web Vulnerabilities

Buffer-overflow attacks occur when an application tries to insert data into an available buffer without checking the size of the buffer that can result in overflow of the buffer [10]. Attackers insert their malicious code into adjacent areas of the stack or heap by inserting larger values that overflow the buffer. These malicious scripts can subsequently be executed by an attacker with the privileges of the Web-server user.

Attackers can launch input validation attacks when the user input is inadequately validated and sanitized. Format string attacks can occur when application code attempts to display unfiltered user-passed variables. Canonicalization attacks occur when equivalent forms of canonical file names are handled differently by a Web-server with unexpected results. Sample applications or test scripts that were often badly coded and were not intended for production use include early Apache test-cgi scripts (CVE1999-0070).

Encoding attacks exploit incorrect handling of various forms of character encoding by the Web server that allows attackers to bypass regular expression filters and launch attacks. Attackers can exploit application or operating system (OS) behavior to execute code with higher than expected privileges resulting in privilege escalation attack. Form-tampering attacks can be launched by client-side manipulation of hidden form fields or modification of browser cookies through local application proxies that alter Web application behavior. In a user-generated content attack, back-doors or malicious server are uploaded into vulnerable Web applications to assist in attacks.

Attackers are sometimes able to manipulate direct references to internal application objects such as file names and user IDs. Insecure direct object reference attacks can include changing an account ID in a dynamically generated Web page from an application that subsequently fails to check if the attacker's user ID is associated with changed account ID.

RFI attacks occur when an application improperly trusts user submitted files or references to external objects such as remote URLs and then evaluate and execute the malicious contents. These are common in PHP applications.

CSRF occurs when an attacker is able to trick an already authenticated user into performing malicious action, which may succeed due to design weakness in the target application and the user's Web browser automatically supplies cached credentials.

Access control weaknesses occur when developers fail to programmatically enforce strict access control allowing attackers to perform unauthorized actions or view restricted material.

Authentication and session management failures result from poorly implemented systems that allow attackers to obtain credentials or tokens to impersonate valid users. Data confidentiality failures occur when user credentials and other confidential data are not protected by correctly implemented proven cryptographic standards. Failure to correctly handle unexpected errors can crash an application revealing internal configuration information or bypass security systems leading to attacks that result from poor error handling.

SQL injection and XSS are the most popular injection attacks. A Web application is vulnerable to SQL injection attacks when malicious content can flow into SQL queries without being fully sanitized allowing the attacker to trigger malicious SQL operations by injecting SQL keywords or operators [11]. Malicious SQL statements can be introduced into a vulnerable application using different input mechanisms including user inputs, cookies, and server variables [12]. Second-order SQL injection attack is a special case, where the attacker stores malicious content into the database and triggers its execution at a later time. These attacks can lead to authentication bypass, information disclosure and other problems.

2.1.2 XSS

Dynamic Web applications accept input from users and perform actions based on the input. Lack of proper input validation results in XSS, which was first discovered in 1990s. It is one of the most common type of injection attacks that exploit input validation vulnerability [13], in which malicious scripts are injected into otherwise benign and trusted Websites [14].

Initially, only two primary types of XSS attacks were identified: Stored XSS (persistent) and Reflected XSS (non-persistent). *Stored XSS*, also referred to as *Persistent XSS*, generally occurs when user input is stored on the target server, such as in a database, message forum, visitor log, or comment field. Subsequently, a victim retrieves the stored data from the Web application without that data being made safe to render in the browser. With the advent of HTML5 and other browser technologies, the attack payload can be permanently stored in the victim's browser, such as an HTML5 database and never sent to the server at all [15]. *Reflected XSS*, also referred to as *Non-Persistent XSS*, occurs when user input is immediately returned by a Web application in an error message, search result, or any other response. The message or response may include some or all of the input provided by the user as part of the request without that data being made safe to render in the browser and without permanently storing the user provided data [15].

Amit Klein [16] identified and defined a third type of XSS, which is termed as *Document Object Model (DOM) Based XSS*. In this type of XSS, the entire tainted data flow from source to

sink takes place in the browser, i.e., the source and sink of the data is in the DOM and the data flow never leaves the browser. For instance, the source could be the URL of the page or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data.

A fourth type of XSS, which was identified and defined in 2007, is known as *Induced XSS* [17]. XSS of this type can be launched when the Webserver has HTTP Response splitting vulnerability. An attacker can change the entire HTML content by manipulating the HTTP header of the server's response to include a request parameter that has not been validated.

The four categories of XSS attacks overlap, i.e., it is possible to have both Stored and Reflected DOM-Based XSS, as well as Stored and Reflected Non-DOM Based XSS. Around mid-2012, the OWASP (Open Web Application Security Project) with involvement of the cyber-security research community proposed two new terms to organize and classify the types of XSS attacks that can occur: Server XSS and Client XSS [15].

Server XSS occurs when untrusted user supplied data is included in an HTML response generated by the server. The source of this data could be from the request, or from a stored location. This leads to two types of server XSS: *Reflected Server XSS* and *Stored Server XSS*. In this case, the entire vulnerability is in server-side code, and the browser is simply rendering the response and executing any valid script embedded in it [15].

Client XSS occurs when untrusted user supplied data is used to update the DOM with an unsafe JavaScript call. A JavaScript call is considered unsafe if it can be used to introduce valid JavaScript into the DOM. The source of this data could be from the DOM, or it could have been sent by the server (via an AJAX call, or a page load). The ultimate source of the data could have been from a request, or from a stored location on the client or the server that leads to two types of client XSS: *Reflected Client XSS* and *Stored Client XSS* [15].

With these new definitions, DOM-Based XSS simply becomes a subset of Client XSS, where the source of the data is somewhere in the DOM, rather than from the Server. The induced

XSS would also fit into this classification proposed by OWASP [15]. It would be a subset of Reflected Server XSS.

To launch any type of aforementioned XSS-attacks there are three generic steps that need to occur in sequential order: locate vulnerability, inject attack script, and execute attack script [18]. With XSS, every input and output has the potential to be an attack vector, which does not occur with other vulnerability types. In addition, XSS has numerous subtleties and variants which makes its detection and/or prevention difficult.

2.2 TECHNIQUES AND METHODS FOR DETECTING AND PREVENTING XSS VULNERABILITIES

Fig. 2.2 presents a categorization and classification that is based on a thorough and systematic review of literature on techniques and methods for detecting and/or preventing XSS vulnerabilities. We classify the work into three broad categories: server-side, client-side, and generic that addresses the work that that is geared towards detecting and/or preventing server-side, client-side or in general XSS-attacks. The work that did not specify whether the approach was geared towards server-side or client-side is classified under a generic category. Each of these categories are further refined into stored, reflected and general type of XSS attacks, which are further refined into approaches for detecting, preventing, or supplementary. Each of the approaches are further refined into whether they are used for detecting or preventing vulnerabilities or attacks. Further refinement is based on whether static analysis, dynamic analysis, modeling, secure programming, or other technique is used to detecting or preventing vulnerabilities and attacks. Static analysis involves reviewing the source code or byte code of an application to find faults. Dynamic analysis entails examining the behavior of an application in runtime. Secure Programming involves using libraries that take care of security attributes or following programming guidelines or rules and practices for secure development of Web applications. Modeling involves using different modeling notations for detection or prevention of XSS related vulnerabilities and attacks. The approaches that did not fall into any of the above mentioned