# MOTH: A HYBRID THREAT MODEL FOR IMPROVING SOFTWARE SECURITY TESTING

HABEEB OLADAPO OMOTUNDE

A thesis submitted in
fulfillment of the requirement for the award of the
Doctor of Philosophy

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

JULY 2018

# DEDICATION

To Almighty Allah and my lovely parents, Tajudeen and Adiat Omotunde.

## ACKNOWLEDGEMENT

All praises, thanks and adorations are due to Almighty Allah for making this journey a success. I am more than grateful to Him for seeing me through this course. I am deeply indebted to my supervisor Prof. Dr. Rosziati Ibrahim whose help, stimulating suggestions, useful critiques and encouragement helped me in all the times of my study under her tutelage. I am deeply indebted to the Office of Research, Innovation, Commercialization and Consultancy Management (ORICC) for sponsoring this research under the Vot No. U193. My utmost gratitude goes to my parents and siblings. You are indeed a blessing to me. Thank you for believing in me. Your motivation, love and support can never be repaid except by Allah in manifolds. Special appreciation to my Mum, Mrs Adiat Olaseni Omotunde, my wife and lovely sons, Abdullah and AbdurRahman. You are my joy and all I have. Thanks for your understanding, encouragement and prayers during the course of this research. I owe you a lot for your support and co-operation.

## ABSTRACT

As SQL injection attack (SQLIA) continues to threaten web applications despite several techniques recommended to prevent it, a Hybrid Threat Modeling strategy was adopted in this research due to its proactive approach to risk mitigation in web applications. This involved the combination of 3 threat modeling techniques namely misuse cases, attack trees and finite state machines in order to harness their individual strengths to design a Hybrid Threat Modeling framework and tool called MOTH (Modeling Threats using Hybrid techniques). Using the MOTH tool developed using Eclipse rich client platform, experimental results with an e-commerce web application downloaded from GitHub namely BodgeIt store shows an improved SQL injection vulnerability detection rate of 13.33% in comparison to a commercial tool, IBM AppScan. Further benchmarking of MOTH with respect to SQL injection vulnerability detection in both BodgeIT store and IBM's Altoro Mutual online banking application shows it is 30.6% more effective over AppScan. Relative to other threat modeling tools, MOTH was able to realize a 41.7% optimization of attack paths required to design effective test plans and test cases for the recommendation of efficient security requirements needed to prevent SQL injection attacks. A 100% risk mitigation was achieved after applying these recommendations due to a complete security test coverage of all test cases during the experiment as all test cases successfully exposed the inherent security mutants in the AUT. These results show that MOTH is a more suitable hybrid threat modeling tool for preventing poor specifications that expose web applications to SQL injection attacks.

# ABSTRAK

Serangan SQL Injection (SQLIA) sering terjadi dan memberi kesan kepada aplikasi-aplikasi web walaupun pelbagai teknik telah dicadangkan untuk mengelakkan ia berlaku. Oleh itu, strategi Hybrid Threat Modeling telah dilaksanakan di dalam kajian ini kerana ia memiliki pendekatan proaktif untuk mengurangkan risiko serangan SQLIA di dalam aplikasi web. Kajian ini telah menggabungkan kelebihan-kelebihan yang terdapat di dalam 3 teknik threat modeling iaitu misuse cases, attack trees dan finite state machines untuk menghasilkan Hybrid Threat Modeling framework dan MOTH (Modeling Threat using Hybrid techniques) tool. MOTH tool telah dibangunkan menggunakan platform Eclipse dan hasil keputusan ekperimen menggunakan aplikasi web e-dagang, BodgeIt yang dimuat turun dari GitHub menunjukkan teknik yang dicadangkan mampu mengesan serangan SQL Injection dengan lebih baik sebanyak 13.33% berbanding tool komersial, IBM AppScan. MOTH juga berupaya mengesan serangan SQL Injection dengan lebih baik sebanyak 30.6% berbanding AppScan bagi aplikasi BodgeIt dan aplikasi perbankan dalam talian, Altoro Mutual IBM. Berbanding dengan threat modeling tools yang lain, MOTH juga mampu mengoptimumkan risiko serangan SQL injection sebanyak 41.7%. 100% pengurangan risiko telah berjaya dicapai selepas mengaplikasikan teknik MOTH. Ini disebabkan oleh liputan ujian keselamatan yang lengkap bagi semua test cases di dalam semua eksperimen dan MOTH berjaya mendedahkan security mutants yang wujud di dalam AUT. Keputusan ini menunjukkan bahawa MOTH adalah hybrid threat modeling tool yang lebih baik dalam mencegah serangan SQL injection.

# CONTENTS

**TITLE** i

**DECLARATION** ii

**DEDICATION** iii

**ACKNOWLEDGEMENT** iv

**ABSTRACT** v

**ABSTRAK** vi

**CONTENTS** vii

**LIST OF TABLES** xii

**LIST OF FIGURES** xiv

**LIST OF ALGORITHMS** xvii

**LIST OF SYMBOLS AND ABBREVIATIONS** xviii

**LIST OF APPENDICES** xx

**LIST OF PUBLICATIONS** xxi

**CHAPTER 1 INTRODUCTION** **1**

1.1 Background Study 1
1.2 Research Motivation 4
1.3 Problem Statement 6
1.4 Research Objectives 7
1.5 Research Scope 8
1.6 Significance of Study 9
1.7 Thesis Outline 10

**CHAPTER 2 LITERATURE REVIEW** **12**

2.1 Introduction 12
2.2 Overview of Software Security Testing 12
2.3 Software Security Testing Techniques 13
2.3.1 Manual Inspection & Reviews 14
2.3.2 Code Review 14
2.3.3 Penetration Testing 16
2.3.4 Threat Modeling 18
2.4 Threat Modeling Techniques 24
2.4.1 Misuse Cases 24
2.4.2 Attack Trees 26
2.4.3 Finite State Machines 28
2.5 Overview of the Hybrid Threat Modeling Approach 31
2.6 Related Works on Hybrid Threat Modeling 32
2.7 Threat Focus: SQL Injection Attacks (SQLIAs) 35
2.7.1 Tautology SQLIA 40
2.7.2 Union-Based SQLIA 42
2.7.3 Boolean-Based Blind SQLIA 43
2.8 Chapter Summary 46

**CHAPTER 3 RESEARCH METHODOLOGY** **47**

3.1 Introduction 47
3.2 Research Process 47
3.3 Research Framework 50
3.4 Input Stage 52
3.4.1 Newly Developed Systems 52
3.4.2 Existing Systems 53
3.5 Activities Stage 53
3.5.1 Phase 1 - Web Application Decomposition and Detection of SQLIVs 53
3.5.2 Phase 2 - Vulnerability Exploitation in Software Assets 55
3.5.3 Hybrid Threat Model Design 57
3.5.4 Phase 3 - Vulnerability Resolution 63
3.6 Output Stage 71
3.6.1 Intermediate Output - SAEV 71
3.6.2 Final Output- Evaluation of MOTH 72
3.7 Chapter Summary 73

**CHAPTER 4 DESIGN AND IMPLEMENTATION OF THE MOTH FRAMEWORK 74**

4.1 Introduction 74
4.2 MOTH Design 75
4.2.1 Eclipse RCP for Plug-in Development 77
4.2.2 Eclipse EMF 78
4.2.3 Eclipse GEF 79
4.2.4 Eclipse GMF 79
4.2.5 MOTH Architecture 79
4.3 Hybrid Threat Model Definition and Design 80
4.3.1 Web Application 81
4.3.2 Attack Trees 81
4.3.3 SQL Injection 82
4.3.4 Non-deterministic Finite Automata (NFA) 83
4.3.5 Modeling Attack Trees as NFA 85
4.4 Implementation Of MOTH Framework 97
4.4.1 MOTH's UI Components and Features 97
4.4.2 SeaMonster Application 98
4.4.3 Diagram Editor Workbench Advisor 99
4.4.4 Diagram Editor Workbench Window Advisor 99
4.4.5 Diagram Editor Perspective 100
4.4.6 Diagram Editor ActionBar Advisor 101
4.5 MOTH Algorithms 102
4.5.1 VulnScan Algorithm 102
4.5.2 BSM Builder Algorithm 103
4.5.3 Analyze Tree Model Algorithm 108
4.5.4 Merge Machine Algorithm 110
4.6 Security Testing 111
4.7 Experimental Set-up 114
4.7.1 Data Setup 114
4.7.2 Server and Tool Setup 115
4.8 Chapter Summary 115

CHAPTER 5 RESULT,ANALYSIS, EVALUATION OF MOTH AND DISCUSSION 116

5.1 Introduction 116
5.2 Case Studies 116
5.2.1 BodgeIT Store 117
5.2.2 Altoro Mutual 121
5.3 Hybrid Threat Model Design 123
5.3.1 State Machine Modeling and Optimization 124
5.3.2 Attack Tree Modeling and Optimization 125
5.4 Security Testing 127
5.4.1 Test Case Generation (TCG) 128
5.4.2 Test Case Execution (TCE) 133
5.4.3 Test Report and Security Requirement Specification 133
5.4.4 Applying Security Recommendations 135
5.4.5 Verification of Security Improvement 139
5.5 Evaluation of MOTH 141
5.5.1 Evaluation of Vulnerability Detection Rate (VDR) 141
5.5.2 Evaluation of Risk Mitigation 143
5.5.3 Evaluation of Security Test Coverage 143
5.6 Comparison of MOTH with Other Threat Modeling Tools 144
5.6.1 Use of Threat Modeling for Software Security Testing 145
5.7 Threat Modeling for Security Requirements Elicitation 147
5.8 Comparison based on Threat Model Development 147
5.9 Chapter Summary 149

CHAPTER 6 CONCLUSION 150

6.1 Research Summary 150
6.2 Achievement of Objectives 151
6.3 Contribution of the Study 153
6.4 Limitation of the Study 153
6.5 Recommendations for Future Work 154

**REFERENCES** **155**

**APPENDICES** **173**

**CURRICULUM VITAE** **187**

## LIST OF TABLES


| | | |
|---|---|---|
| 2.1 | Threat Modelling Research Activities | 23 |
| 2.2 | User Authentication Module Vulnerabilities and Security Specifications (Khan, 2015) | 29 |
| 2.3 | Hybrid Threat Modeling Researches | 32 |
| 2.4 | Differences between Misuse Cases and Attack Trees | 33 |
| 2.5 | Top 10 Most Frequently Exploited Categories of Websites | 36 |
| 3.1 | Misuse Case Template | 59 |
| 3.3 | Security Mutants in Magento Study | 65 |
| 3.4 | Test Case Design Template for login.jsp | 67 |
| 4.1 | MOTH dependency Table on SeaMonster Modeling Tool | 76 |
| 4.2 | Tree Implementation Matrix of Tautology Attack Tree to NFAs | 87 |
| 4.3 | Tree Implementation Matrix of Union Based Attack Tree to NFAs | 94 |
| 4.4 | Tree Implementation Matrix of Boolean Based Blind Attack Tree to NFAs | 96 |
| 4.5 | Mapping Security Tests to the Vulnerable Assets | 112 |
| 4.6 | Test Plan for verifying Tautology SQLIA in Vulnerable Asset - login.jsp | 113 |
| 5.1 | Overview of bodgeit store web application modules | 117 |
| 5.2 | Distribution of SQL injection Vulnerability across the AUT | 118 |
| 5.3 | Vulnerable SQL Statements in Bodgeit store | 120 |
| 5.4 | SQLIVs detected with MOTH in Altoro Mutual | 123 |
| 5.5 | SQLIVs detected with AppScan in Altoro Mutual | 123 |
| 5.6 | State Machine Optimization for all SQL statement | 125 |
| 5.7 | Attack Path Optimization of Attack Trees | 126 |
| 5.8 | Mapping Attack Paths to Vulnerable Assets | 128 |
| 5.9 | Deriving Test Cases for Login.jsp | 129 |
| 5.10 | Deriving Test Cases for Basket and Advanced.jsp | 129 |
| 5.11 | Deriving Test Cases for Register and Password.jsp | 129 |
| 5.12 | Test Cases 1, 2 and 3 for Login.jsp | 130 |

| | | |
|---|---|---|
| 5.13 | Designing Test Cases 4,5 and 7 for login.jsp using SQL Map | 130 |
| 5.14 | Designing Test Case 6 for Login.jsp | 130 |
| 5.15 | Test Cases 8 and 9 for Basket.jsp | 131 |
| 5.16 | Test Case 10 for Advanced.jsp | 132 |
| 5.17 | Test Cases 11 and 12 for Register.jsp | 132 |
| 5.18 | Test Case 13 for Password.jsp | 133 |
| 5.19 | Test Case Report and Security Recommendations | 135 |
| 5.20 | Recommendations categorized into Classes | 136 |
| 5.21 | Redesigned SQL Statements as Parameterized Queries | 137 |
| 5.22 | Comparison of Test Case Results before and After application of Security requirements | 141 |
| 5.23 | Tool Comparison | 142 |
| 5.24 | Number of SQLIVs detected by MOTH and AppScan | 142 |
| 5.25 | Comparison Based on Test Case Generation and Execution for Software Security Testing | 145 |
| 5.26 | Comparison Based on Guide to Hybrid Threat Modeling Design | 148 |

## LIST OF FIGURES


| | | |
|---|---|---|
| 1.1 | Hybrid Threat Modeling at Early Phases of the SSDLC | 8 |
| 2.1 | Cost of fixing software defects over time | 17 |
| 2.2 | The Secure Software development Life cycle | 19 |
| 2.3 | The Microsoft Threat Modeling Process | 23 |
| 2.4 | A Misuse Case representation of user account spoofing attack | 25 |
| 2.5 | An Attack Tree representation of user account spoofing attack | 27 |
| 2.6 | State Machine Representation of Security Specifications for the User Authentication Module (Khan, 2015) | 29 |
| 2.7 | Surge in Breach Rate and Types of private information disclosed | 36 |
| 2.8 | SQL Injection Categories | 38 |
| 2.9 | JSPFirst Login Form | 38 |
| 2.10 | New User Registration Form | 39 |
| 2.11 | Valid and invalid log on | 39 |
| 2.12 | Tautology Based SQL Injection log on successful | 41 |
| 2.13 | Malicious query reveals number of DB columns | 43 |
| 2.14 | Malicious query reveals database name and type | 44 |
| 2.15 | Generic Error when Boolean Based Blind SQLIA fails | 45 |
| 2.16 | Extracting Data with Boolean Based Blind SQLIA | 45 |
| 3.1 | Research Process Flow Chart | 48 |
| 3.2 | Summary of the MOTH framework | 50 |
| 3.3 | MOTH Hybrid Threat Modeling Framework | 51 |
| 3.4 | Mapping the Research Process to the MOTH Framework | 52 |
| 3.5 | Relationship between Risk Management and Test planning | 54 |
| 3.6 | Phase 1 Steps and Output | 54 |
| 3.7 | Phase 2 Steps and Output | 55 |
| 3.8 | Misuse Case Modeling for SQLIA | 58 |
| 3.9 | Tautology SQLIA Attack Tree Model | 61 |
| 3.10 | Merging SQL Injection Misuse Cases with Attack Tree Goals | 61 |
| 3.11 | Authentication Module as a Finite State Machine | 62 |

| | | |
|---|---|---|
| 3.12 | Phase 3 Steps | 63 |
| 3.13 | Selenium WebDriver Architecture | 68 |
| 3.14 | Eclipse Set-Up for Security Test Execution | 69 |
| 3.15 | Replacing Dynamic Queries with Prepared Statement | 70 |
| 4.1 | Developing MOTH using Eclipse RCP Integrated Development Environment | 77 |
| 4.2 | MOTH Security Model | 80 |
| 4.3 | MOTH Architectural Diagram | 80 |
| 4.4 | SQLIA Attack Tree Model | 81 |
| 4.5 | LogInStateMachine $(L_{sm})$ | 84 |
| 4.6 | Tautology SQLIA Attack Tree Model | 85 |
| 4.7 | Tautology Attack Tree Modelled as a set of NFAs | 88 |
| 4.8 | Simulating Tautology SQLIA via an Attack Vector | 90 |
| 4.9 | Login.jsp | 90 |
| 4.10 | Union Based SQLIA Attack Tree Model | 92 |
| 4.11 | Union Based Attack Tree Modelled as a set of NFAs | 93 |
| 4.12 | Boolean Based Blind SQLIA Attack Tree Model | 95 |
| 4.13 | Boolean Based Blind Attack Tree Modelled as a set of NFAs | 95 |
| 4.14 | SeaMonster Application | 98 |
| 4.15 | Application Workbench Advisor | 99 |
| 4.16 | MOTH Window Configuration | 100 |
| 4.17 | MOTH Layout Configuration with Perspective | 101 |
| 4.18 | Overview of a Workbench window and its parts | 101 |
| 4.19 | MOTH Hybrid Threat Modeling Tool | 102 |
| 4.20 | Optimized State Machine Model of Original and Malicious SQL Statement | 104 |
| 4.21 | BSMBuilder result after optimization (optimizedAutomata.gv) | 107 |
| 4.22 | Node and their properties | 109 |
| 4.23 | An Optimized Attack Path Detection using MOTH's Analyze Tree Model Algorithm | 109 |
| 4.24 | Software Asset With Exploitable Vulnerability | 111 |
| 4.25 | Simulating Security Testing with Attack Path, $AP_{t1}$ | 114 |
| 4.26 | Deploying Bodgeit store Web application for security testing | 115 |
| 5.1 | Bodgeit Store Use Case Diagram | 118 |
| 5.2 | Scanning BodgeIT for vulnerable SQL statements with MOTH | 119 |
| 5.3 | Distribution of vulnerable SQL statements across Bodgeit | 119 |
| 5.4 | Result after Scanning BodgeIt store with AppScan | 121 |

| | | |
|---|---|---|
| 5.5 | Result after Scanning BodgeIt store with AppScan | 124 |
| 5.6 | Optimization of State Machines | 125 |
| 5.7 | Attack Path Optimization | 126 |
| 5.8 | TCE with Selenium Before Applying Security Requirements | 134 |
| 5.9 | TCE Report Before implementing Security Requirements | 134 |
| 5.10 | Input Validation | 138 |
| 5.11 | Cookie Type Validation | 138 |
| 5.12 | Using a least privilege account for database transactions | 139 |
| 5.13 | TCE with Selenium After Applying Security Requirements | 140 |
| 5.14 | Vulnerability Assessment with SQL Injection Detection Tools | 143 |
| 5.15 | Risk Mitigation after Applying Security Recommendations | 144 |

# LIST OF ALGORITHMS


| NO. | TITLE | PAGE |
|---|---|---|
| 1 | VulnScan (SQL injection Vulnerability Detection Algorithm) | 103 |
| 2 | BSMBuilder (Build an Optimized State Machine Model) | 105 |
| 3 | Analyze Tree Model (Attack Path Detection and Optimization algorithm using DFS Technique) | 108 |
| 4 | Formation of SAEV | 110 |
| 5 | Security Testing Algorithm | 112 |

## LIST OF SYMBOLS AND ABBREVIATIONS

| | | |
|---|---|---|
| AUT | – | Application Under Test |
| BNF | – | Backus Naur Form |
| CAPEC | – | Common Attack Pattern Enumeration and Classification |
| CERN | – | Center for European Nuclear Research |
| CI5A | – | Confidentiality, Integrity, Availability, Authentication, Authorization, Accounting, and Anonymity |
| CVE | – | Common Vulnerabilities and Exposures |
| DAG | – | Directed Acyclic Graphs |
| DAST | – | Dynamic Application Security Testing |
| DFS | – | Depth First Search |
| DBMS | – | Database Management System |
| DOS | – | Denial of Services |
| EMF | – | Eclipse Modeling Framework |
| GEF | – | Graphics Editing Framework |
| GMF | – | Graphics Modeling Framework |
| GOAT | – | Graphical Overview and Analysis Tool |
| HTM | – | Hybrid Threat Modeling |
| HTML | – | Hypertext Mark-up Language |
| HTTP | – | Hypertext Transfer Protocol |
| IAST | – | Interactive Application Security Testing |
| IDE | – | Integrated Development Environment |
| IMPV | – | Security Improvement |
| MLFA | – | Multi level Automata |
| MOTH | – | Modeling Threats using Hybrid-Techniques |
| NIST | – | National Institute of Standards and Technology |
| OWASP | – | Open Web Application Security Project |
| PDE | – | Plug-in Development Environment |
| pFSM | – | Predicate Finite State Machines |
| SAEV | – | Software Asset with Exploitable Vulnerability |
| SAST | – | Static Application Security Testing |
| SDLC | – | Software Development Life-Cycle |
| SOAP | – | Simple Object Access Protocol |
| SQL | – | Structured Query Language |
| SQLIA | – | SQL Injection Attack |
| SQLIV | – | SQL Injection Vulnerability |
| SSDL | – | Secure Software Development Life-Cycle |
| STRIDE | – | Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of privilege |

| | | |
|---|---|---|
| SVRS | – | Security Vulnerability Repository Service |
| TP | – | Test Plan |
| TCE | – | Test Case Execution |
| TCG | – | Test Case Generation |
| VDR | – | Vulnerability Detection Rate |
| WASC | – | Web Application Security Consortium |
| XMI | – | XML Metadata Interchange |
| XML | – | Extensible Markup Language |

## LIST OF APPENDICES


| APPENDIX | TITLE | PAGE |
|---|---|---|
| A | Some Code Listings for MOTH | 173 |
| B | Deploying MOTH | 183 |
| C | Misuse Case Template | 184 |

# LIST OF PUBLICATIONS

## JOURNALS:

I **Habeeb Omotunde**, Rosziati Ibrahim and Maryam Ahmed (2017): ”An Optimized Attack Tree Model for Security Test Case Planning and Generation”. In Journal of Theoretical and Applied Information Technology Vol. 96 issue 17. (Indexed by Scopus).

II **Habeeb Omotunde**, Rosziati Ibrahim, Maryam Ahmed, Rasheedah Olanrewaju, Noraini Ibrahim and Habeeb Shah (2016): ”A Framework to Reduce Redundancy in Android Test Suite using Refactoring”. In Indian Journal of Science and Technology Vol. 9 issue 46. (Indexed by Scopus)

III **Habeeb Omotunde**, Rosziati Ibrahim (2015): ”A Review of Threat Modeling & Its Hybrid Approaches to Software Security Testing”. Paper presented at the 4th International Conference on Research and Innovation in Information Systems (ICRIIS) 2015 International Conference in Melaka, Malaysia. Published in ARPN Journal of Engineering and Applied SciencesVol. 10 No. 23 (Indexed by Scopus)

## CONFERENCE PROCEEDINGS:

I **Habeeb Omotunde**, Rosziati Ibrahim. (2016). ”A Hybrid Threat Model for Software Security Requirement Specification.” Paper presented at the 3rd International Conference on Information Science and Security (ICISS) 2016 International conference in Pattaya, Thailand. Published in IEEE Xplore Digital Library (Indexed by Scopus).

II **Habeeb Omotunde**, Rosziati Ibrahim(2015). ”Mitigating SQL Injection (SQLi) Attacks Via Hybrid Threat Modelling”. Paper presented at the 2nd International Conference on Information Science and Security (ICISS) 2015 International conference in Seoul, South Korea. Published in IEEE Xplore Digital Library (Indexed by Scopus).

# CHAPTER 1

# INTRODUCTION

## 1.1 Background Study

As organizations seek to fulfil their objectives in the 21st century, they have come to immensely depend on reliable and secure software as a core component of their organizational asset to achieve their set goals (Symantec, 2014; Amthor *et al.*, 2014). These software assets are system resources that have significant value to the stakeholders of the organization (Wichers and Williams, 2013). Irrespective of the size, nature or sector of these organizations, securing the software asset has gained momentum (Johnson *et al.*, 2013; Zhang *et al.*, 2014) given the explosion of software vulnerabilities (Sultana *et al.*, 2017) leading to major software security issues in the form of incessant cyber-attacks to confidential data or mission critical systems which could bring huge losses to both the organization and her customers (Kavitha *et al.*, 2017; Pacheco *et al.*, 2017). These kinds of attacks include but are not limited to SQL injection, denial of service, disclosure of confidential information and data theft or corruption via social engineering attacks, phishing attacks, watering hole attacks, buffer overflow or stack smashing (Pickard *et al.*, 2012; Bozic *et al.*, 2013; Chen *et al.*, 2013; Marback *et al.*, 2013; Shar and Tan, 2013). These could push organizations out of business due to customers' lack of trust in using the services, mitigating laws enacted by the government or legal issues raised by aggrieved parties for breach of contract (Paul, 2014).

However, post deployment and reactive measures such as software patching and upgrade, damage assessment, logging analysis, installation of intrusion detection and prevention systems to mention a few have not stopped or deterred attackers from continuously bombarding these software assets using more sophisticated attacks to exploit the software vulnerabilities (Kar and Panigrahi, 2013; Li *et al.*, 2017). These myriads of unending threats have prompted software security experts to propose proactive strategies of building security into the traditional Software Development Life Cycle (SDLC) hence the Secure Software Development Lifecycle (SSDL) paradigm came to life (OWASP, 2014a; Tatli, 2018). Given the unique culture and practices of disparate IT firms, many tech giants have thrown their weight behind the creation of proprietary software security models such as Trustworthy Computing Secure Development Lifecycle from Microsoft (Microsoft, 2005), CLASP (Comprehensive Lightweight Security Application Process) and Open SAMM (Software Assurance Maturity Model) from OWASP (Open Web Application Security Project) (OWASP, 2016) and Touchpoints from Cigital (McGraw, 2006). Interestingly, over 60 tech-fortune companies such as SONY, VISA, Intel, Microsoft etc. have collaborated to develop a descriptive framework tagged BSIMM (Building Security In Maturity Model) (BSIMM, 2014). These new paradigms and the aggressive allocation of resources (funds and man-power) to such projects have empowered the development and security team to address security issues during the earliest stages of system development (Karpati *et al.*, 2014). In the secure software development lifecycle, one of the critical approaches to defending the organizations software infrastructure is to anticipate the nature of the attacks from the attacker's perspective before they happen and strategizing mitigation plans in order to prevent these attacks from being successful. This is called Threat Modeling (Groves, 2013).

Threat modeling is a software security practice utilized by software developers, architects and security experts at the design phase of software development to document the key assets found in a software application and intentionally expose those assets to security risks in a thorough and disciplined manner. The goal of a threat modeling exercise is to detect hidden software vulnerabilities regarded as

”entry points” (Shostack, 2014) that may elude the application developers and use this information to develop mitigation strategies thereby providing a roadmap for proactive security plans (SecurityInnovation, 2011).

By identifying an application’s potential vulnerabilities, threat modeling helps the development and security team to understand and prioritize the array of risks for which these discovered vulnerabilities are susceptible in the event of an attack. With the results of a threat model at hand, development teams can ensure that they are concentrating their design, implementation or testing efforts on the risks that matter most considering the direct or indirect impact of such risks on the business (SecurityInnovation, 2011). In a nutshell, identifying threats during the threat modelling exercise helps software security engineers come up with realistic and valuable security requirements (Myagmar *et al.*, 2005). These security requirements are constraints that govern the intended behaviour of a software application in accordance with the security goals and policies set by the organization (Haley *et al.*, 2008). Therefore, threat modeling is vital for software vulnerability detection and prevention.

Given the above premises, researchers have proposed many methods for developing threat models such as the use of attack trees (Swideski and Snider, 2004), threat nets (Dianxiang *et al.*, 2012) a formal specification method adapted from Petri Nets, use of sequence diagrams to monitor possible threats during program execution (Wang *et al.*, 2007), finite state machines for modeling software objects behavior (Chen *et al.*, 2003) and Misuse cases, a variation of the UML Use Case model (Sindre and Opdahl, 2005a). In the field of software security testing, this approach has also been used by Wang (Wang *et al.*, 2007) and Dianxiang (Dianxiang *et al.*, 2012) to test for software security in the design phase of the software development. Marback *et al.* (2013) successfully applied attack trees to generate security test cases which might help in identifying threats capable of compromising security.

## 1.2 Research Motivation

Over the years, many researchers have taken threat modeling a step further by experimentally comparing these modeling techniques especially attack trees and misuse cases. This was done in order to discover the possibility of combining them as an hybrid for complementary use or rather substitute them as alternatives (Opdahl and Sindre, 2009).

One of the earliest Hybrid Threat Modeling (HTM) tools developed by a community of researchers in the academic and industry to resolve software security issues was SeaMonster (Meland *et al.*, 2008). It was created in order to bridge the communication gap between security experts and software developers as a means to enhance knowledge sharing about software vulnerabilities. Misuse case and attack tree threat models were used in SeaMonster to connect different aspect of every detected vulnerability so as to understand the causes of these vulnerabilities, threats liable to exploit them and mitigation strategies to prevent their successful exploitation (Meland *et al.*, 2008).

These two techniques were chosen by many researchers because they both focused mainly on what the attacker is trying to achieve, and in turn provide mitigation strategies to foil the attack (Karpati *et al.*, 2014; Mai *et al.*, 2018). An experiment was performed by Opdahl and Sindre (2009) using software engineering students to measure effectiveness, coverage, perceived usefulness, perceived ease of use and intention to use of both threat modeling techniques i.e. Attack trees and Misuse Cases. Although, the result showed that attack trees, when compared to misuse cases, were more efficient in identifying threats particularly those related to confidentiality and authorization, however, manual inspection of the experimental results indicated that both techniques are complementary to an extent (Opdahl and Sindre, 2009). Further experiments were needed to clarify the complementary nature of these techniques hence Karpati *et al.* (2014) embarked on an experiment to compare attack trees and misuse cases in an industrial setting taking his experimental and control group from

# REFERENCES

Acunetix (2017). *OWASP Top 10 Critical Web App Vulnerabilities*. Retrievable at https://www.acunetix.com/vulnerability-scanner/scan-website-owasp-top-10-risks/.

Ahmed, M. and Ibrahim, R. (2015). A Comparative study of web application testing and mobile application testing. *Lecture Notes in Electrical Engineering*. 315, 491–500. doi:10.1007/978-3-319-07674-4_48. Retrievable at https://www.scopus.com/inward/record.uri?eid=2-s2.0-84915750469.

Al-Azzani, S. (2014). *Architecture-centric testing for security*. Ph.D. Thesis. University of Birmingham.

Amthor, P., Kühnhauser, W. E. and Pölck, A. (2014). WorSE: a workbench for model-based security engineering. *Computers & Security*. 42, 40–55.

Antunes, N. and Vieira, M. (2009). Detecting SQL injection vulnerabilities in web services. In *Dependable Computing, 2009. LADC 09. Fourth Latin-American Symposium on*. IEEE, 17–24.

Antunes, N. and Vieira, M. (2014). Penetration Testing for Web Services. *Computer*. 47(2), 30–36. Antunes, Nuno Vieira, Marco.

Avancini, A. (2012). Security Testing of Web Applications: A Research Plan. In *34th International Conference on Software Engineering (ICSE)*. International Conference on Software Engineering. 1491–1494. Avancini, Andrea.

Baca, D., Carlsson, B., Petersen, K. and Lundberg, L. (2013). Improving software security with static automated code analysis in an industry setting. *Software: Practice and Experience*. 43(3), 259–279. ISSN 1097-024X. doi:10.1002/spe.2109. Retrievable at http://dx.doi.org/10.1002/spe.2109.

Banerjee, C., Banerjee, A. and Sharma, S. (2017). Estimating influence of threat using Misuse Case Oriented Quality Requirements (MCOQR) metrics:

Security requirements engineering perspective. *International Journal of Hybrid Intelligent Systems*. 14(1-2), 1–11. ISSN 1448-5869.

Banerjee, C., Poonia, A. S., Banerjee, A. and Sharma, S. K. (2018). Proposed Algorithm for Identification of Vulnerabilities and Associated Misuse Cases Using CVSS, CVE Standards During Security Requirements Elicitation Phase. In *Soft Computing: Theories and Applications*. Springer. ISBN 978-981-10-5699-4, 651–658.

Batool, S. and Asghar, S. (2014). Secure State UML: Modeling and Testing Security Concerns of Software Systems Using UML State Machines. *Research Journal of Applied Sciences, Engineering and Technology*. 7(18), 3786–3790.

Bennetts, S. (2016). *The BodgeIt Store*. Retrievable at https://github.com/psiinon/bodgeit.

Blome, A., Ochoa, M., Li, K. Q., Peroli, M., Dashti, M. T. and Soc, I. C. (2013). VERA: A flexible model-based vulnerability testing tool. *2013 Ieee Sixth International Conference on Software Testing, Verification and Validation (Icst 2013)*, 471–478. doi:10.1109/icst.2013.65. Retrievable at ⟨GotoISI⟩://WOS:000332473300053.

Bozic, J., Wotawa, F. and Ieee (2013). XSS Pattern for Attack Modeling in Testing. *2013 8th International Workshop on Automation of Software Test (Ast)*, 71–74. Retrievable at ⟨GotoISI⟩://WOS:000332877400012.

Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M. and Pretschner, A. (2005). *Model-based testing of reactive systems: advanced lectures*. vol. 3472. Springer.

BSIMM (2014). *Building Security In Maturity Model*. Retrievable at http://www.bsimm.com.

CERN (2016). *Static Code Analysis Tools*. Retrievable at https://security.web.cern.ch/security/recommendations/en/code_tools.shtml.

Chandrashekhar, R., Mardithaya, M., Thilagam, S. and Saha, D. (2012). SQL Injection Attack Mechanisms and Prevention Techniques. In Thilagam, P., Pais, A., Chandrasekaran, K. and Balakrishnan, N. (Eds.) *Advanced Computing,* *Networking and Security*. (pp. 524–533). *Lecture Notes in Computer Science*, vol. 7135. Springer Berlin Heidelberg. ISBN 978-3-642-29279-8. doi: 10.1007/978-3-642-29280-4_61. Retrievable at http://dx.doi.org/10.1007/978-3-642-29280-4_61.

Chen, L.-H., Hsu, F.-H., Hwang, Y., Su, M.-C., Ku, W.-S. and Chang, C.-H. (2013). ARMORY: An automatic security testing tool for buffer overflow defect detection. *Computers Electrical Engineering*. 39(7), 2233–2242. ISSN 0045-7906; 1879-0755. doi:10.1016/j.compeleceng.2012.07.005. Retrievable at ⟨GotoISI⟩://WOS:000326661600025.

Chen, S., Kalbarczyk, Z., Xu, J. and Iyer, R. K. (2003). A data-driven finite state machine model for analyzing security vulnerabilities. In *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*. 605–614. doi:10.1109/DSN.2003.1209970.

Chernak, Y. (2001). Validating and Improving Test-Case Effectiveness. *IEEE software*. 18(1), 81–86.

Choras, M., Kozik, R., Puchalski, D. and Holubowicz, W. (2013). *Correlation Approach for SQL Injection Attacks Detection*, *Advances in Intelligent Systems and Computing*, vol. 189. 177–185. Retrievable at ⟨GotoISI⟩://WOS:000312969500018.

Christensen, A. S., Moller, A. and Schwartzbach, M. I. (2003). Precise Analysis of String Expressions. In *International Static Analysis Symposium*. Springer, 1–18.

Das, D., Sharma, U. and Bhattacharyya, D. K. (2017). Defeating SQL injection attack in authentication security: an experimental study. *International Journal of Information Security*. ISSN 1615-5270. doi:10.1007/s10207-017-0393-x. Retrievable at https://doi.org/10.1007/s10207-017-0393-x.

Deng, M., Wuyts, K., Scandariato, R., Preneel, B. and Joosen, W. (2011). A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*. 16(1), 3–32.

Dhakkan, D. (2013). *SQL Injections: An Introduction*. Retrievable at http://resources.infosecinstitute.com.

Dharam, R. and Shiva, S. G. (2013). Runtime Monitors to Detect and Prevent Union Query based SQL Injection Attacks. Premier Hall Sci & Engn. ISBN 978-0-7695-4967-5, 357–362. doi:{10.1109/ITNG.2013.57}. 10th International Conference on Information Technology - New Generations (ITNG), Las Vegas, NV, APR 15-17, 2013.

Dianxiang, X., Manghui, T., Sanford, M., Thomas, L., Woodraska, D. and Weifeng, X. (2012). Automated Security Test Generation with Formal Threat Models. *Dependable and Secure Computing, IEEE Transactions on*. 9(4), 526–540. ISSN 1545-5971. doi:10.1109/tdsc.2012.24.

Diaz, G. and Bermejo, J. R. (2013). Static analysis of source code security: Assessment of tools against SAMATE tests. *Information and Software Technology*. 55(8), 1462–1476. Diaz, Gabriel Ramon Bermejo, Juan.

Dukes, L., Yuan, X. H., Akowuah, F. and Ieee (2013). A Case Study on Web Application Security Testing with Tools and Manual Testing. In *IEEE SoutheastCon*. IEEE SoutheastCon-Proceedings. Dukes, LaShanda Yuan, Xiaohong Akowuah, Francis.

El-Attar, M. (2012). Towards developing consistent misuse case models. *Journal of Systems and Software*. 85(2), 323–339. ISSN 0164-1212.

Felderer, M., Buchler, M., Johns, M., Brucker, A., Breu, R. and Pretschner, A. (2016). Security Testing: A Survey. *Advances in Computers*. 101, 1–51. doi:10.1016/bs.adcom.2015.11.003. Retrievable at https://www.scopus.com/inward/record.uri?eid=2-s2.0-84952037753&doi=10.1016%2fbs.adcom.2015.11.003&partnerID=40&md5=3a7344a1956972cc81c879187d5e089e.

Fernandez, E. B., Alder, E., Bagley, R. and Paghdar, S. (2012). A Misuse Pattern for Retrieving Data from a Database Using SQL Injection. In *BioMedical Computing (BioMedCom), 2012 ASE/IEEE International Conference on*. 127–131. doi:10.1109/BioMedCom.2012.27.

Firesmith, D. (2007). Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *Journal of Object Technology*. 6(1), 17–33.

Ford, R., Carvalho, M., Mayron, L., Bishop, M. and Ieee (2013). Antimalware Software: Do we Measure Resilience? *Proceedings of the 2013 Ieee Workshop on Anti-Malware Testing Research (Water'13)*, 17–23. Retrievable at ⟨GotoISI⟩://WOS:000332987700003.

Friedman, G., Hartman, A., Nagin, K. and Shiran, T. (2002). Projected state machine coverage for software testing. *SIGSOFT Softw. Eng. Notes*. 27(4), 134–143. ISSN 0163-5948. doi:10.1145/566171.566192.

Gandotra, V., Singhal, A. and Bedi, P. (2009). Identifying Security Requirements Hybrid Technique, 407–412. doi:10.1109/icsea.2009.65.

Garn, B., Kapsalis, I., Simos, D. E. and Winkler, S. (2014). On the applicability of combinatorial testing to web application security testing: a case study. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*. ACM. ISBN 1450329330, 16–21.

Granlund, H. (2009). *Integration of SVRS into the modelling tool GOAT*.

Groves, D. (2013). Application Threat Modelling. *Open Web Application Security Project (OWASP)*, 1–18. Retrievable at https://www.owasp.org/index.php?title=Application_Threat_Modeling\&oldid=146761.

Gruber, H. and Holzer, M. (2008). Finite automata, digraph connectivity, and regular expression size. In *International Colloquium on Automata, Languages, and Programming*. Springer, 39–50.

Haley, C., Laney, R., Moffett, J. and Nuseibeh, B. (2008). Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*. 34(1), 133–153. ISSN 0098-5589. doi:10.1109/TSE.2007.70754.

Halfond, W., Viegas, J. and Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*. IEEE, 65–81.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*. 8(3), 231–274.

Hong, Y., Liu, X. M., Huang, S., Zheng, C. Y. and Ieee (2012). Data Oriented Software Security Testing. *Proceedings of the 2012 Second International Conference on Instrumentation Measurement, Computer, Communication and Control (Imccc 2012)*, 676–679. doi:10.1109/imccc.2012.164. Retrievable at ⟨GotoISI⟩://WOS:000324691100158.

Horner, M. and Hyslip, T. (2017). SQL Injection: The Longest Running Sequel in Programming History. *Journal of Digital Forensics, Security and Law*. 12(2), 10.

Hui, Z. W., Huang, S., Liu, X. M. and Hu, B. (2012). An Integrated Model for Software Security Testing Requirements Behavior. *Information-an International Interdisciplinary Journal*. 15(11A), 4435–4442. ISSN 1343-4500. Retrievable at ⟨GotoISI⟩://WOS:000311066200016.

Hummel, O. and Burger, S. (2017). Analyzing source code for automated design pattern recommendation. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Analytics*. ACM, 8–14.

IBM (2018). *Altoro Mutual Online Banking Application*. Retrievable at http://msdn.microsoft.com/en-us/library/ms995349.aspx.

Jang, Y.-S. and Choi, J.-Y. (2014). Detecting SQL injection attacks using query result size. *Computers Security*. 44(0), 104–118. ISSN 0167-4048. doi:http://dx.doi.org/10.1016/j.cose.2014.04.007. Retrievable at http://www.sciencedirect.com/science/article/pii/S0167404814000595.

Johnson, R., Wang, Z. H., Stavrou, A., Voas, J. and Ieee (2013). Exposing Software Security and Availability Risks For Commercial Mobile Devices. *59th Annual*

*Reliability and Maintainability Symposium (Rams)*. Retrievable at ⟨GotoISI⟩://WOS:000321693500120.

Junqua, J.-C. and van Noord, G. (2001). *Robustness in Language and Speech Technology*. MIT Press.

Jurgenson, A. (2010). *Efficient Semantics of Parallel and Serial Models of Attack Trees*. TUT Press.

Just, R. (2014). The Major Mutation Framework: Efficient and Scalable Mutation Analysis for Java. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ISSTA 2014. New York, NY, USA: ACM. ISBN 978-1-4503-2645-2, 433–436. doi:10.1145/2610384.2628053. Retrievable at http://doi.acm.org/10.1145/2610384.2628053.

Just, R., Jalali, D. and Ernst, M. D. (2014). Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ISSTA 2014. New York, NY, USA: ACM. ISBN 978-1-4503-2645-2, 437–440. doi:10.1145/2610384.2628055. Retrievable at http://doi.acm.org/10.1145/2610384.2628055.

Kacker, R. N., Kuhn, D. R., Lei, Y. and Lawrence, J. F. (2013). Combinatorial testing for software: An adaptation of design of experiments. *Measurement*. 46(9), 3745–3752. ISSN 0263-2241. doi:10.1016/j.measurement.2013.02.021. Retrievable at ⟨GotoISI⟩://WOS:000324298700089.

Kar, D. and Panigrahi, S. (2013). Prevention of SQL Injection Attack Using Query Transformation and Hashing. *Proceedings of the 2013 3rd Ieee International Advance Computing Conference (Iacc)*, 1317–1323. ISSN 2164-8263. Retrievable at ⟨GotoISI⟩://WOS:000321780700237.

Karpati, P., Redda, Y., Opdahl, A. L. and Sindre, G. (2014). Comparing attack trees and misuse cases in an industrial setting. *Information and Software Technology*. 56(3), 294–308.

Karpati, P., Sindre, G. and Opdahl, A. L. (2010). Towards a Hacker Attack Representation Method.

Katkalov, K., Moebius, N., Stenzel, K., Borek, M. and Reif, W. (2014). Modeling test cases for security protocols with SecureMDD. *Computer Networks*. 58, 99–111. ISSN 1389-1286. doi:10.1016/j.comnet.2013.08.024. Retrievable at ⟨GotoISI⟩://WOS:000331781500009.

Kaur, N. and Kaur, P. (2014). Mitigation of SQL Injection Attacks Using Threat Modeling. *SIGSOFT Softw. Eng. Notes*. 39(6), 1–6. ISSN 0163-5948. doi:10.1145/2674632.2674638. Retrievable at http://doi.acm.org/10.1145/2674632.2674638.

Kavitha, C., Gillian, C., Orla, C., Hon, L., Benjamin, N., Brigid, O. G., Dick, O., Scott, W., Paul, W., Candid, W. *et al.* (2017). Symantec Internet Security Threat Report 2017. *Volume XXII*.

Khalil, H. and Labiche, Y. (2017). State-Based Tests Suites Automatic Generation Tool (STAGE-1). In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. ISBN 0730-3157, 357–362. doi:10.1109/COMPSAC.2017.221.

Khamaiseh, S. and Xu, D. (2017). Software Security Testing via Misuse Case Modeling. In *IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing*. 534–541.

Khan, M. U. (2015). Representing Security Specifications in UML State Machine Diagrams. *Procedia Computer Science*. 56, 453–458.

Kordy, B., Pietre Cambacedes, L. and Schweitzer, P. (2014). DAG-based attack and defense modeling: Dont miss the forest for the attack trees. *Computer science review*. 13, 1–38.

Lam, P., Bodden, E., Lhotak, O. and Hendren, L. (2011). The Soot framework for Java program analysis: a retrospective. In *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, vol. 15. 35.

Li, C., Li, M., Liu, S. and Nakajima, S. (2013). *Applying Functional Scenario-Based Test Case Generation Method in Unit Testing and Integration Testing*, Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 7787, book section 1. ISBN 978-3-642-39276-4, 1–11. doi:10.1007/978-3-642-39277-1_1. Retrievable at http://dx.doi.org/10.1007/978-3-642-39277-1_1.

Li, W., Zhang, Z. and Wang, L. (2017). Improvement in diversify active defense for web application by using language and database heterogeneity. In *2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. 30–35. doi:10.1109/ICASID.2017.8285738.

Liu, B. C., Shi, L., Cai, Z. H., Li, M. and Ieee (2012). Software Vulnerability Discovery Techniques: A Survey. In *4th International Conference on Multimedia Information Networking and Security (MINES)*. International Conference on Multimedia Information Networking and Security. 152–156. Liu, Bingchang Shi, Liang Cai, Zhuhua Li, Min.

Liu, L., Su, G., Xu, J., Zhang, B., Kang, J., Xu, S., Li, P. and Si, G. (2017). An Inferential Metamorphic Testing Approach to Reduce False Positives in SQLIV Penetration Test. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. ISBN 0730-3157, 675–680. doi:10.1109/COMPSAC.2017.276.

LiU, S. and Rios, E. (2008). D2. 2 Initial Modelling Methods and Prototype Modelling Tools.

Madhuri, K., Suman, M., Sri, M. N., Kumar, K. R. and Kameswari, U. J. (2012). A Systematic Approach to Generate and Conduct Destructive Security Test Sets. *Mems, Nano and Smart Systems, Pts 1-6*. 403-408, 4495–4498. ISSN 1022-6680. doi:10.4028/www.scientific.net/AMR.403-408.4495. Retrievable at ⟨GotoISI⟩://WOS:000310764702143.

Mai, P. X., Goknil, A., Shar, L. K., Pastore, F., Briand, L. C. and Shaame, S. (2018). Modeling Security and Privacy Requirements: a Use Case-Driven Approach. *Information and Software Technology*. ISSN 0950-5849. doi:https://doi.org/

10.1016/j.infsof.2018.04.007. Retrievable at http://www.sciencedirect.com/science/article/pii/S0950584918300703.

Mammar, A., Mallouli, W. and Cavalli, A. (2012). A systematic approach to integrate common timed security rules within a TEFSM-based system specification. *Information and Software Technology*. 54(1), 87–98.

Marback, A., Do, H., He, K., Kondamarri, S. and Xu, D. (2013). A threat model-based approach to security testing. *Software-Practice and Experience*. 43(2), 241–258.

Matuleviius, R., Norta, A. and Samartel, S. (2018). Security Requirements Elicitation from Airline Turnaround Processes. *Business Information Systems Engineering*. 60(1), 3–20. ISSN 1867-0202. doi:10.1007/s12599-018-0518-4. Retrievable at https://doi.org/10.1007/s12599-018-0518-4.

McAffer, J., Lemieux, J.-M. and Aniszczyk, C. (2010). *Eclipse Rich Client Platform*. Addison-Wesley Professional.

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley Professional. ISBN 0321356705.

McWhirter, P. R., Kifayat, K., Shi, Q. and Askwith, B. (2018). SQL Injection Attack classification through the feature extraction of SQL query strings using a Gap-Weighted String Subsequence Kernel. *Journal of Information Security and Applications*. 40, 199–216. ISSN 2214-2126. doi:https://doi.org/10.1016/j.jisa.2018.04.001. Retrievable at http://www.sciencedirect.com/science/article/pii/S2214212617303691.

Meland, P., Spampinato, D. G., Hagen, E., Baadshaug, E. T., Krister, K.-M. and Velle, K. S. (2008). SeaMonster: Providing tool support for security modeling. *Norsk informasjonssikkerhetskonferanse, NISK*.

Meland, P., Tndel, I. and Jensen, J. (2010). *Idea: Reusability of Threat Models Two Approaches with an Experimental Evaluation*, Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 5965, book section 9.

Meland, P. H., Ardi, S., Jensen, J., Rios, E., Sanchez, T., Shahmehri, N. and Tondel, I. A. (2009). An architectural foundation for security model sharing and reuse. In *International Conference on Availability, Reliability and Security, 2009. ARES'09*. IEEE, 823–828.

Menzel, M., Thomas, I. and Meinel, C. (2009). Security Requirements Specification in Service-Oriented Business Process Management. In *International Conference on Availability, Reliability and Security, 2009. ARES '09*. March. 41–48. doi: 10.1109/ARES.2009.90.

Michael, C. C. and Radosevich, W. (2005). Risk-Based and Functional Security Testing. *Build Security In*.

Microsoft (2005). *The Trustworthy Computing Security Development Lifecycle*. Retrievable at http://msdn.microsoft.com/en-us/library/ms995349.aspx.

Mirembe, D. P. and Muyeba, M. (2008). Threat Modeling Revisited: Improving Expressiveness of Attack. In *Computer Modeling and Simulation, 2008. EMS '08. Second UKSIM European Symposium on*. 93–98.

MITRE (2014). *Common Vulnerabilities and Exposures*. Retrievable at https://cve.mitre.org/.

Mittal, P., Jena, S. K. and Ieee (2013). *A Fast and Secure Way to Prevent SQL Injection Attacks*. 2013 Ieee Conference on Information and Communication Technologies. ISBN 978-1-4673-5758-6; 978-1-4673-5759-3. Retrievable at ⟨GotoISI⟩://WOS:000325208700140.

Mumtaz, H., Alshayeb, M., Mahmood, S. and Niazi, M. (2018). An empirical study to improve software security through the application of code refactoring. *Information and Software Technology*. 96, 112–125. ISSN 0950-5849. doi:https://doi.org/10.1016/j.infsof.2017.11.010. Retrievable at http://www.sciencedirect.com/science/article/pii/S0950584916303664.

Myagmar, S., Lee, A. J. and Yurcik, W. (2005). Threat Modeling as a Basis for Security Requirements. In *Symposium on requirements engineering for information security (SREIS)*, vol. 2005. Citeseer, 1–8.

Natarajan, K. and Subramani, S. (2012). Generation of Sql-injection Free Secure Algorithm to Detect and Prevent Sql-Injection Attacks. *Procedia Technology*. 4(0), 790–796. ISSN 2212-0173. doi:http://dx.doi.org/10.1016/j.protcy.2012.05.129. Retrievable at http://www.sciencedirect.com/science/article/pii/S2212017312004082.

NIST (2016). *Source Code Security Analyzers*. Retrievable at https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html.

Opdahl, A. L. and Sindre, G. (2009). Experimental comparison of attack trees and misuse cases for security threat identification. *Information and Software Technology*. 51(5), 916–932. ISSN 0950-5849. doi:http://dx.doi.org/10.1016/j.infsof.2008.05.013. Retrievable at http://www.sciencedirect.com/science/article/pii/S0950584908000773.

Oracle (2017). *MySQL 5.7 Reference Manual and Documentation*. Retrievable at https://dev.mysql.com/doc/refman/5.7/en/union.html.

Ouchani, S., Mohamed, O. A., Debbabi, M. and Pourzandi, M. (2010). Verification of the correctness in composed UML behavioural diagrams. In *Software Engineering Research, Management and Applications 2010*. (pp. 163–177). Springer.

OWASP (2013). *Blind SQL Injection*. Retrievable at https://www.owasp.org/index.php/Blind_SQL_Injection.

OWASP (2014a). *Testing Guide Introduction*. Retrievable at https://www.owasp.org/index.php/Testing_Guide_Introduction.

OWASP (2014b). Threat Risk Modeling. *Open Web Application Security Project (OWASP)*.

OWASP (2015). *Testing for SQL Injection*. Retrievable at https://www.owasp.org/index.php.

OWASP (2016). *OWASP CLASP Project*. Retrievable at https://www.owasp.org/index.php/Category:OWASP_CLASP_Project.

Pacheco, J., Ibarra, D., Vijay, A. and Hariri, S. (2017). IoT Security Framework for Smart Water System. In *ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. 1285–1292. doi:10.1109/AICCSA.2017.85.

Paul, M. (2014). *Software Security: Being Secure in an Insecure World [White paper]*. Retrievable at https://www.isc2.org.

Pickard, C., Miladinov, S. and Ieee (2012). Rogue software: Protection against potentially unwanted applications. *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software*, 1–8. Retrievable at ⟨GotoISI⟩://WOS:000318854300001.

Raspotnig, C., Karpati, P. and Opdahl, A. L. (2018). Combined Assessment of Software Safety and Security Requirements: An Industrial Evaluation of the CHASSIS Method. *Journal of Cases on Information Technology (JCIT)*. 20(1), 46–69.

Redhat (2017). *How Threat Modeling Helps Discover Security Vulnerabilities*. Retrievable at https://access.redhat.com/blogs/766093/posts/2914051.

Rosen, K. H. (2012). Discrete mathematics and its applications. *Amc*. 10(12), 824.

Sadeghian, A., Zamani, M. and Abd Manaf, A. (2013a). A Taxonomy of SQL Injection Detection and Prevention Techniques. ISBN 978-0-7695-5133-3, 53–56. doi:{10.1109/ICICM.2013.18}. International Conference on Informatics and Creative Multimedia (ICICM), Kuala Lumpur, Malaysia, Sep 04-06, 2013.

Sadeghian, A., Zamani, M., Ibrahim, S. and Ieee (2013b). SQL Injection is Still Alive: A Study on SQL Injection Signature Evasion Techniques. *2013 International Conference on Informatics and Creative Multimedia (Icicm)*, 265–268. doi:10.1109/icicm.2013.52. Retrievable at ⟨GotoISI⟩://WOS:000343826000050.

Saidane, A. and Guelfi, N. (2013). *Towards test-driven and architecture model-based security and resilience engineering*. 163–188. doi:10.4018/978-1-4666-2958-5.ch010.

Salman, Y. D., Hashim, N. L., Rejab, M. M., Romli, R. and Mohd, H. (2017). Coverage criteria for test case generation using UML state chart diagram. In *AIP Conference Proceedings*, vol. 1891. AIP Publishing. ISBN 0735415730, 020125.

Salva, S. and Regainia, L. (2017). Using Data Integration for Security Testing. In Yevtushenko, N., Cavalli, A. R. and Yenign, H. (Eds.) *Testing Software and Systems*. Springer International Publishing. ISBN 978-3-319-67549-7, 178–194.

Saxena, A., Sengupta, S., Duraisamy, P., Kaulgud, V., Chakraborty, A. and Ieee (2013). *Detecting SOQL-Injection Vulnerabilities in SalesForce Applications*. 2013 International Conference on Advances in Computing, Communications and Informatics. ISBN 978-1-4799-2432-5; 978-1-4799-2659-6. Retrievable at ⟨GotoISI⟩://WOS:000343771500083.

Schneier, B. (1999). Attack trees. *Dr. Dobbs journal*. 24(12), 21–29.

SecurityFocus (2014). *bugtraq database*. Retrievable at http://www.securityfocus.com/archive/1.

SecurityInnovation (2011). *Threat Modelling for Secure Embedded Software [White paper]*. Retrievable at http://web.securityinnovation.com/threat-modeling-embedded/.

SecurityInnovation (2014). *Static code analysis is better at the desktop [White paper]*. Retrievable at http://www.klocwork.com.

Selenium (2016). *SeleniumHQ Test Automation*. Retrievable at http://www.seleniumhq.org/.

Shahriar, H., North, S. and Chen, W. C. (2013). *Client-Side Detection of SQL Injection Attack*, *Lecture Notes in Business Information Processing*, vol. 148. 512–517. Retrievable at ⟨GotoISI⟩://WOS:000345280400046.

Shanmughaneethi, V. and Swamynathan, S. (2012). Detection of SQL Injection Attack in web applications using web services. *IOSR Journal of Computer Engineering (IOSRJCE)*. 1(5), 13–20.

Shar, L. K. and Tan, H. B. K. (2013). Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Information and Software Technology*. 55(10), 1767–1780. ISSN 0950-5849. doi:http://dx.doi.org/10.1016/j.infsof.2013.04.002. Retrievable at http://www.sciencedirect.com/science/article/pii/S0950584913000852.

Shostack, A. (2008). Experiences threat modeling at microsoft. In *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*.

Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley. ISBN 9781118809990. Retrievable at http://books.google.com.my/books?id=asPDAgAAQBAJ.

Sindre, G., Firesmith, D. G. and Opdahl, A. L. (2003). A reuse-based approach to determining security requirements. In *Proceedings of the 9th international workshop on requirements engineering: foundation for software quality (REFSQ 03), Klagenfurt, Austria*. Citeseer.

Sindre, G. and Opdahl, A. L. (2005a). Eliciting security requirements with misuse cases. *Requirements Engineering*. 10(1), 34–44.

Sindre, G. and Opdahl, A. L. (2005b). Eliciting Security Requirements With Misuse Cases. *Requirements Engineering*. 10(1), 34–44. ISSN 0947-3602. doi:10.1007/s00766-004-0194-4. Retrievable at ⟨GotoISI⟩://WOS:000226269900003.

Sipser, M. (2006). *Introduction to the Theory of Computation*. vol. 2. Thomson Course Technology Boston.

Sneed, H. M. (2004). Measuring the Effectiveness of Software Testing. *SOQUA TECOS*. 58, 109–120.

Soni, M. (2014). *Defect Prevention: Reducing Costs and Enhancing Quality*. Retrievable at http://www.isixsigma.com.

Stackoverflow (2017). *Difference Between Union and Union All*. Retrievable at http://stackoverflow.com/questions/49925/what-is-the-difference-between-union-and-union-all.

Su, Z. and Wassermann, G. (2006). The Essence of Command Injection Attacks in Web Applications. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '06. New York, NY, USA: ACM. ISBN 1-59593-027-2, 372–382. doi:10.1145/1111037.1111070. Retrievable at http://doi.acm.org/10.1145/1111037.1111070.

Subramani, S., Vouk, M., Williams, L. and Ieee (2013). Non-Operational Testing of Software for Security Issues. *2013 Ieee International Symposium on Software Reliability Engineering Workshops (Issrew)*, 21–22. Retrievable at ⟨GotoISI⟩://WOS:000330639500011.

Sultana, K. Z., Williams, B. J. and Bhowmik, T. (2017). A study examining relationships between micro patterns and security vulnerabilities. *Software Quality Journal*. ISSN 1573-1367. doi:10.1007/s11219-017-9397-z. Retrievable at https://doi.org/10.1007/s11219-017-9397-z.

Swideski, F. and Snider, W. (2004). *Threat Modeling*. Microsoft Press. ISBN 0735619913.

Symantec (2014). *Web Security Threat Report*.

Talukder, A. K., Maurya, V. K., Santhosh, B. G., Jangam, E., Muni, S. V., Jevitha, K. P., Saurabh, S. and Pais, A. R. (2009). Security-aware Software Development Life Cycle (SaSDLC) - Processes and tools. In *Wireless and Optical Communications Networks, 2009. WOCN '09. IFIP International Conference on*. 1–5.

Tatli, E. s. (2018). Developer-oriented Web Security by Integrating Secure SDLC into IDEs. *Sakarya University Journal of Computer and Information Sciences*. 1(1), 36–43.

Tndel, I. A., Jaatun, M. G., Cruzes, D. S. and Moe, N. B. (2017). Risk Centric Activities in Secure Software Development in Public Organisations. *International Journal of Secure Software Engineering (IJSSE)*. 8(4), 1–30.

Tondel, I. A., Jensen, J. and Rostad, L. (2010). Combining misuse cases with attack trees and security activity models. *Fifth International Conference on*

*Availability, Reliability, and Security: Ares 2010, Proceedings*, 438–445.

Trusted-Consultant (2007). *Secure Software Engineering and Risk Management Strategies for Building Secure Web Applications*. Retrievable at http://securesoftware.blogspot.my/2007/02/justin-schuh-question-3.html.

Tuma, K., Scandariato, R., Widman, M. and Sandberg, C. (2017). Towards Security Threats that Matter. In *Computer Security*. (pp. 47–62). Springer.

Van den Berghe, A., Yskout, K. and Joosen, W. (2018). Security patterns 2.0: Towards security patterns based on security building blocks. In *SEAD18: IEEE/ACM 1st International Workshop on Security Awareness from Design to Deployment*. ACM.

Wang, L., Wong, E. and Xu, D. (2007). A threat model driven approach for security testing. In *Software Engineering for Secure Systems, 2007. SESS'07: ICSE Workshops 2007. Third International Workshop on*. IEEE, 10–10.

Washizaki, H. (2017). Security patterns: Research direction, metamodel, application and verification. In *2017 International Workshop on Big Data and Information Security (IWBIS)*. 1–4. doi:10.1109/IWBIS.2017.8275094.

Wichers, D. and Williams, J. (2013). OWASP Top 10 Most Critical Web Application Security Risks. *OWASP Foundation*.

Williams, I. and Yuan, X. (2017). Creating Abuse Cases Based on Attack Patterns: A User Study. In *2017 IEEE Cybersecurity Development (SecDev)*. 85–86. doi:10.1109/SecDev.2017.27.

Wu, T.-Y., Chen, C.-M., Sun, X., Liu, S. and Lin, J. C.-W. (2017). A Countermeasure to SQL Injection Attack for Cloud Environment. *Wireless Personal Communications*. 96(4), 5279–5293. ISSN 1572-834X. doi:10.1007/s11277-016-3741-7. Retrievable at https://doi.org/10.1007/s11277-016-3741-7.

Wysopal, C., Nelson, L., Dustin, E. and Zovi, D. (2007). *The Art of Software Security Testing: Identifying Software Security Flaws*. Addison-Wesley. ISBN 9780321304865.

Yadav, N. and Shekokar, N. (2018). Analysis on Injection Vulnerabilities of Web Application. In Vasudevan, H., Deshmukh, A. A. and Ray, K. P. (Eds.) *Proceedings of International Conference on Wireless Communication*. Springer Singapore. ISBN 978-981-10-8339-6, 13–22.

Zech, P., Felderer, M. and Breu, R. (2017). Knowledge-based security testing of web applications by logic programming. *International Journal on Software Tools for Technology Transfer*. ISSN 1433-2787. doi:10.1007/s10009-017-0472-3. Retrievable at https://doi.org/10.1007/s10009-017-0472-3.

Zhang, D. Z., Liu, D. G., Csallner, C., Kung, D. and Lei, Y. (2014). A distributed framework for demand-driven software vulnerability detection. *Journal of Systems and Software*. 87, 60–73. ISSN 0164-1212. doi:10.1016/j.jss.2013.08.033. Retrievable at ⟨GotoISI⟩://WOS:000329273400005, zhang, Dazhi Liu, Donggang Csallner, Christoph Kung, David Lei, Yu.

Zhang, J., Zhang, L., Harman, M., Hao, D., Jia, Y. and Zhang, L. (2018). Predictive mutation testing. *IEEE Transactions on Software Engineering*.

Zuo, C., Zhao, Q. and Lin, Z. (2017). AUTHSCOPE: Towards Automatic Discovery of Vulnerable Authorizations in Online Services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 799–813.