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.
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 mengoptimalkan 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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE</td>
<td>i</td>
</tr>
<tr>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td>vi</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF ALGORITHMS</td>
<td>xvii</td>
</tr>
<tr>
<td>LIST OF SYMBOLS AND ABBREVIATIONS</td>
<td>xviii</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td>xx</td>
</tr>
<tr>
<td>LIST OF PUBLICATIONS</td>
<td>xxi</td>
</tr>
</tbody>
</table>

## CHAPTER 1 INTRODUCTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Background Study</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Research Motivation</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Problem Statement</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Research Objectives</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Research Scope</td>
<td>8</td>
</tr>
<tr>
<td>1.6 Significance of Study</td>
<td>9</td>
</tr>
<tr>
<td>1.7 Thesis Outline</td>
<td>10</td>
</tr>
</tbody>
</table>
CHAPTER 2 LITERATURE REVIEW

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 SQUIVs 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

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

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

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
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

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Hybrid Threat Modeling at Early Phases of the SSDLC</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Cost of fixing software defects over time</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>The Secure Software development Life cycle</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>The Microsoft Threat Modeling Process</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>A Misuse Case representation of user account spoofing attack</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>An Attack Tree representation of user account spoofing attack</td>
<td>27</td>
</tr>
<tr>
<td>2.6</td>
<td>State Machine Representation of Security Specifications for the User Authentication Module (Khan, 2015)</td>
<td>29</td>
</tr>
<tr>
<td>2.7</td>
<td>Surge in Breach Rate and Types of private information disclosed</td>
<td>36</td>
</tr>
<tr>
<td>2.8</td>
<td>SQL Injection Categories</td>
<td>38</td>
</tr>
<tr>
<td>2.9</td>
<td>JSPFirst Login Form</td>
<td>38</td>
</tr>
<tr>
<td>2.10</td>
<td>New User Registration Form</td>
<td>39</td>
</tr>
<tr>
<td>2.11</td>
<td>Valid and invalid log on</td>
<td>39</td>
</tr>
<tr>
<td>2.12</td>
<td>Tautology Based SQL Injection log on successful</td>
<td>41</td>
</tr>
<tr>
<td>2.13</td>
<td>Malicious query reveals number of DB columns</td>
<td>43</td>
</tr>
<tr>
<td>2.14</td>
<td>Malicious query reveals database name and type</td>
<td>44</td>
</tr>
<tr>
<td>2.15</td>
<td>Generic Error when Boolean Based Blind SQLIA fails</td>
<td>45</td>
</tr>
<tr>
<td>2.16</td>
<td>Extracting Data with Boolean Based Blind SQLIA</td>
<td>45</td>
</tr>
<tr>
<td>3.1</td>
<td>Research Process Flow Chart</td>
<td>48</td>
</tr>
<tr>
<td>3.2</td>
<td>Summary of the MOTH framework</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>MOTH Hybrid Threat Modeling Framework</td>
<td>51</td>
</tr>
<tr>
<td>3.4</td>
<td>Mapping the Research Process to the MOTH Framework</td>
<td>52</td>
</tr>
<tr>
<td>3.5</td>
<td>Relationship between Risk Management and Test planning</td>
<td>54</td>
</tr>
<tr>
<td>3.6</td>
<td>Phase 1 Steps and Output</td>
<td>54</td>
</tr>
<tr>
<td>3.7</td>
<td>Phase 2 Steps and Output</td>
<td>55</td>
</tr>
<tr>
<td>3.8</td>
<td>Misuse Case Modeling for SQLIA</td>
<td>58</td>
</tr>
<tr>
<td>3.9</td>
<td>Tautology SQLIA Attack Tree Model</td>
<td>61</td>
</tr>
<tr>
<td>3.10</td>
<td>Merging SQL Injection Misuse Cases with Attack Tree Goals</td>
<td>61</td>
</tr>
<tr>
<td>3.11</td>
<td>Authentication Module as a Finite State Machine</td>
<td>62</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>3.12</td>
<td>Phase 3 Steps</td>
<td>63</td>
</tr>
<tr>
<td>3.13</td>
<td>Selenium WebDriver Architecture</td>
<td>68</td>
</tr>
<tr>
<td>3.14</td>
<td>Eclipse Set-Up for Security Test Execution</td>
<td>69</td>
</tr>
<tr>
<td>3.15</td>
<td>Replacing Dynamic Queries with Prepared Statement</td>
<td>70</td>
</tr>
<tr>
<td>4.1</td>
<td>Developing MOTH using Eclipse RCP Integrated Development Environment</td>
<td>77</td>
</tr>
<tr>
<td>4.2</td>
<td>MOTH Security Model</td>
<td>80</td>
</tr>
<tr>
<td>4.3</td>
<td>MOTH Architectural Diagram</td>
<td>80</td>
</tr>
<tr>
<td>4.4</td>
<td>SQLIA Attack Tree Model</td>
<td>81</td>
</tr>
<tr>
<td>4.5</td>
<td>LogInStateMachine ($L_{sm}$)</td>
<td>84</td>
</tr>
<tr>
<td>4.6</td>
<td>Tautology SQLIA Attack Tree Model</td>
<td>85</td>
</tr>
<tr>
<td>4.7</td>
<td>Tautology Attack Tree Modelled as a set of NFAs</td>
<td>88</td>
</tr>
<tr>
<td>4.8</td>
<td>Simulating Tautology SQLIA via an Attack Vector</td>
<td>90</td>
</tr>
<tr>
<td>4.9</td>
<td>Login.jsp</td>
<td>90</td>
</tr>
<tr>
<td>4.10</td>
<td>Union Based SQLIA Attack Tree Model</td>
<td>92</td>
</tr>
<tr>
<td>4.11</td>
<td>Union Based Attack Tree Modelled as a set of NFAs</td>
<td>93</td>
</tr>
<tr>
<td>4.12</td>
<td>Boolean Based Blind SQLIA Attack Tree Model</td>
<td>95</td>
</tr>
<tr>
<td>4.13</td>
<td>Boolean Based Blind Attack Tree Modelled as a set of NFAs</td>
<td>95</td>
</tr>
<tr>
<td>4.14</td>
<td>SeaMonster Application</td>
<td>98</td>
</tr>
<tr>
<td>4.15</td>
<td>Application Workbench Advisor</td>
<td>99</td>
</tr>
<tr>
<td>4.16</td>
<td>MOTH Window Configuration</td>
<td>100</td>
</tr>
<tr>
<td>4.17</td>
<td>MOTH Layout Configuration with Perspective</td>
<td>101</td>
</tr>
<tr>
<td>4.18</td>
<td>Overview of a Workbench window and its parts</td>
<td>101</td>
</tr>
<tr>
<td>4.19</td>
<td>MOTH Hybrid Threat Modeling Tool</td>
<td>102</td>
</tr>
<tr>
<td>4.20</td>
<td>Optimized State Machine Model of Original and Malicious SQL Statement</td>
<td>104</td>
</tr>
<tr>
<td>4.21</td>
<td>BSMBuilder result after optimization (optimizedAutomata.gv)</td>
<td>107</td>
</tr>
<tr>
<td>4.22</td>
<td>Node and their properties</td>
<td>109</td>
</tr>
<tr>
<td>4.23</td>
<td>An Optimized Attack Path Detection using MOTH’s Analyze Tree Model Algorithm</td>
<td>109</td>
</tr>
<tr>
<td>4.24</td>
<td>Software Asset With Exploitable Vulnerability</td>
<td>111</td>
</tr>
<tr>
<td>4.25</td>
<td>Simulating Security Testing with Attack Path, $AP_{t1}$</td>
<td>114</td>
</tr>
<tr>
<td>4.26</td>
<td>Deploying Bodgeit store Web application for security testing</td>
<td>115</td>
</tr>
<tr>
<td>5.1</td>
<td>Bodgeit Store Use Case Diagram</td>
<td>118</td>
</tr>
<tr>
<td>5.2</td>
<td>Scanning BodgeIT for vulnerable SQL statements with MOTH</td>
<td>119</td>
</tr>
<tr>
<td>5.3</td>
<td>Distribution of vulnerable SQL statements across Bodgeit</td>
<td>119</td>
</tr>
<tr>
<td>5.4</td>
<td>Result after Scanning BodgeIT store with AppScan</td>
<td>121</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.5</td>
<td>Result after Scanning BodgeIt store with AppScan</td>
<td>124</td>
</tr>
<tr>
<td>5.6</td>
<td>Optimization of State Machines</td>
<td>125</td>
</tr>
<tr>
<td>5.7</td>
<td>Attack Path Optimization</td>
<td>126</td>
</tr>
<tr>
<td>5.8</td>
<td>TCE with Selenium Before Applying Security Requirements</td>
<td>134</td>
</tr>
<tr>
<td>5.9</td>
<td>TCE Report Before implementing Security Requirements</td>
<td>134</td>
</tr>
<tr>
<td>5.10</td>
<td>Input Validation</td>
<td>138</td>
</tr>
<tr>
<td>5.11</td>
<td>Cookie Type Validation</td>
<td>138</td>
</tr>
<tr>
<td>5.12</td>
<td>Using a least privilege account for database transactions</td>
<td>139</td>
</tr>
<tr>
<td>5.13</td>
<td>TCE with Selenium After Applying Security Requirements</td>
<td>140</td>
</tr>
<tr>
<td>5.14</td>
<td>Vulnerability Assessment with SQL Injection Detection Tools</td>
<td>143</td>
</tr>
<tr>
<td>5.15</td>
<td>Risk Mitigation after Applying Security Recommendations</td>
<td>144</td>
</tr>
</tbody>
</table>
## LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VulnScan (SQL injection Vulnerability Detection Algorithm)</td>
<td>103</td>
</tr>
<tr>
<td>2</td>
<td>BSMBuilder (Build an Optimized State Machine Model)</td>
<td>105</td>
</tr>
<tr>
<td>3</td>
<td>Analyze Tree Model (Attack Path Detection and Optimization algorithm using DFS Technique)</td>
<td>108</td>
</tr>
<tr>
<td>4</td>
<td>Formation of SAEV</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>Security Testing Algorithm</td>
<td>112</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVRS</td>
<td>Security Vulnerability Repository Service</td>
</tr>
<tr>
<td>TP</td>
<td>Test Plan</td>
</tr>
<tr>
<td>TCE</td>
<td>Test Case Execution</td>
</tr>
<tr>
<td>TCG</td>
<td>Test Case Generation</td>
</tr>
<tr>
<td>VDR</td>
<td>Vulnerability Detection Rate</td>
</tr>
<tr>
<td>WASC</td>
<td>Web Application Security Consortium</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
## LIST OF APPENDICES

<table>
<thead>
<tr>
<th>APPENDIX</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Some Code Listings for MOTH</td>
<td>173</td>
</tr>
<tr>
<td>B</td>
<td>Deploying MOTH</td>
<td>183</td>
</tr>
<tr>
<td>C</td>
<td>Misuse Case Template</td>
<td>184</td>
</tr>
</tbody>
</table>
LIST OF PUBLICATIONS

JOURNALS:


CONFERENCE PROCEEDINGS:


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


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


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


