VOL. 10, NO. 23, DECEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

A REVIEW OF THREAT MODELLING and ITS HYBRID APPROACHES TO SOFTWARE SECURITY TESTING

Habeeb Omotunde and Rosziati Ibrahim
University Tun Hussein Onn Malaysia, Batu Pahat, Malaysia
E-Mail: hi130033@siswa.uthm.edu.my

ABSTRACT

As organizations seek to fulfill 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. Irrespective of the size, nature or sector of these firms, securing the software asset has gained momentum given major software security issues in the form of incessant cyber-attacks to sensitive and confidential data which could bring huge losses to both the organization and her customers. However, a critical approach to defending the organization's software infrastructure is anticipating the nature of the exploits from the attacker's perspective before they occur and strategizing mitigation plans in order to prevent these attacks from being successful. This is called Threat Modeling. The objective of this paper is to identify existing challenges in this research field and establish the grounds for a credible research activity therefore the researchers present a review of literatures on threat modelling activities overs the years and the subsequent hybrids developed to cater for the weaknesses of the techniques used. It was discovered that software applications suffered from analysis paralysis due to over-specification of security requirements while using hybrid threat modeling techniques. Furthermore, we discuss briefly our proposed approach to using hybrid threat modeling using a set of coherent modeling techniques in tackling a particular security vulnerability plaguing web applications while avoiding analysis paralysis.

Keywords: threat modeling, hybrid threat modeling, SSDL, software security, software vulnerability, web applications.

INTRODUCTION

Software vulnerabilities in the form of bugs and flaws are the major gateways to security violations in computer programs deployed by organizations [1]. Given the continuous rise in the cost of fixing these vulnerabilities, legal issues raised by aggrieved partied for breach of contract, customers' lack of trust in using the services and enactment of tough laws by government and much more which makes it difficult for such organizations to run smoothly, software developers and security experts have proposed proactive strategies for building security into the traditional Software Development Life-Cycle(SDLC) hence the Secure Software Development Life-Cycle (SSDL) paradigm came to life [2]. Software development using the SSDL framework not only ensures that the software fulfills the functional requirements, it strictly guarantees the specified security requirements in all development phases by ensuring the software does not do what it was never designed for. This has enabled both teams of software developers and security experts to address software security concerns at the earlier phases of software development including the design phase where threat modeling is executed [3].

By definition, threat modeling is a risk management strategy to proactively secure software assets by anticipating the nature of attacks that could exploit the software vulnerabilities from the attacker's perspective and putting up plans and measures to prevent such attacks from being successful [4]. Having identified 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 impact of such risks on the business [5].

Given the above premises, researchers have proposed many methods for developing threat models such as the use of threat or attack trees [6] which was adapted from Fault Trees in safety analysis, threat nets [7] a formal specification method adapted from Petri Nets, the use of sequence diagrams to monitor possible threats during program execution [8], behavioral state machines for modeling software object's behavior [9] and Misuse cases, a variation of the UML Use Case model [10]. Furthermore, Marback *et al* [11] successfully tested for software security using attack trees to generate security test cases which might help in identifying threats capable of compromising security policies. This approach has also been used by [8, 12] to test for software security in the design phase of the software development.

The rest of this paper is organized as follows: Section 2 briefly provides background information on software security testing and its techniques. Section 3 discusses the methodology of this research work while section 4 dives into details on threat modeling. Section 5 ushers in hybrid threat modelling and the related works by previous researchers while discussion will be given in Section 6 including a brief insight into the proposed hybrid threat modeling algorithm after which we conclude in Section 7.

OVERVIEW OF SOFTWARE SECURITY TESTING TECHNIQUES

Software security testing is a process for validating the secure implementation of a software product

thus reducing the likelihood of a product containing security flaws being released and discovered by customers or malicious users. A major goal of software security testing is to find vulnerabilities, keep them away from the final product, and confirm that the security of the software is at an acceptable level [13].

This process is performed regardless of the type of functionality the software implements. Therefore, while traditional software testing verifies that all use cases and functionalities detailed in the requirements documents are fully implemented according to specification, security testing goes further to ensure that the software does not do what it's not supposed to do [14]. Its function is to assess the security properties and behavior of that software as it interacts with external entities across the trust boundaries [4] (humans, its environment and other software installations) while its own component interact with each other. This is important as the potentially hostile environment could cause security breaches that pose severe consequences [15]. During the testing process, the objectives of security testing team are:

- To ensure a predictable and secure software behaviour
- To ensure that software vulnerabilities remain hidden from third parties.
- To guarantee the maintenance of a secure state via error and exception handling.
- To confirm that all implicit and specified security requirements are satisfied while no security constraints are violated.

It is important to mention that there are several security testing techniques adopted across the development phases by application developers whereby the technique used depends on what phase the software development and security teams are currently working on. Table 1 shows the distribution of the techniques across the software development life-cycle.

Table-1. Software security testing techniques.

#	**Development phase**	**Testing techniques**
1	Requirement Specification	• Misuse Cases • Attack Models
2	Design	• Threat Modeling • Architectural and Design Review • Formal Proof
3	Implementation	• Code Review • Compile Time Detection • Static Analysis Fault Injection
	Similar Techniques in Phases 3 & 4	• Fuzz Testing • Binary Code Analysis • Vulnerability Scanning
4	Verification	• Static Analysis • Source code Fault Injection • Binary Fault Injection • Penetration Testing
5	Deployment & Maintenance	• Static and Impact Analysis • Vulnerability Scanning • Regression testing

Source: United States department of homeland security [13].

Given the focus of this paper, we will move on to discussing about threat modelling and its hybrid techniques. But before that, the method adopted in carry out this research will be highlighted.

METHODOLOGY

This paper provides a literature review of existing challenges in threat modelling techniques, its hybrids and SQL injection using several digital libraries for related publications. The researchers considered relevant electronic databases such as Web of Science, Scopus, Science Direct, IEEE Explorer and Springer Link for studies between 2009 and 2015 inclusive. A total of 101 primary studies including conference proceedings, journals and book chapters were found and critically studied to extract useful information and identify current challenges in the threat modelling field while restricting the considered vulnerability to SQL injection hence narrowing the research scope and search terms. Apart from these repositories, online forums such as the Open Web Application Security Project (OWASP), Security reports from Symantec and MITRE's Common Vulnerabilities and Exposure (CVE) repositories were deeply consulted alongside white papers from leading security firms around the world. Book sources were not left behind so as to have a sound understanding of the topic been discussed.

After reviewing the full texts for identified challenges, contribution, solutions and future works, gaps were identified and the researchers categorized the relevant studies chronologically into years these studies were conducted starting from the most recent. It is important to consider that the threat modelling research field have received far less attention as compared to other software engineering research fields hence the need to cite researches earlier than 2009. In conclusion, a hybrid threat modeling algorithm was presented in section 6, detailing how the researchers intend to avoid over-specification of security requirements while preventing SQL injection in web applications.

THREAT MODELING

Threat modeling provides a systematic way to identify threats that might compromise security, and it has been a well-accepted practice by the industry [11]. It includes determining the attack surface of the software by examining its functionality for trust boundaries, vulnerabilities and poor design specifications [16] .It is to be performed only after the security requirements are complete, so that the threat model is based on the security needs of the software [17]. A vendor neutral SSDL model such as that in Figure 1 must be adopted and tailored by each organization according to her business needs [14].

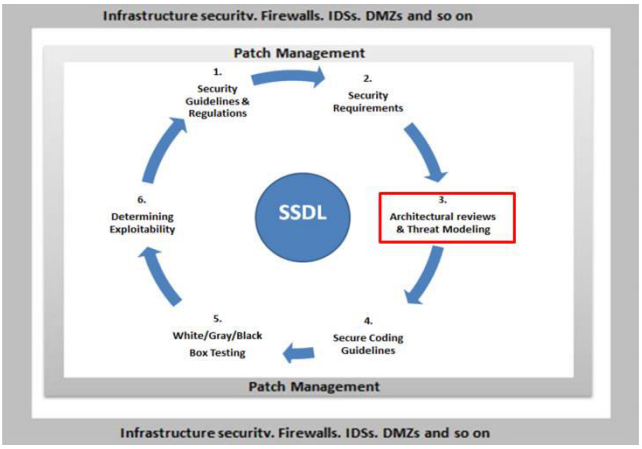

Figure-1. The secure software development life-cycle
Source: [14]

An interesting approach to examining vulnerabilities was undertaken by [9, 18] using finite state machines. This was a stepping stone into the realm of threat modeling. Chen and his team's assertion were to focus on understanding how threats are realized by studying the system vulnerabilities first. Using the Bugtraq database [19], the team selected the most prevalent and exploited vulnerability classes which represented 22% of the total content of the database at that moment to derive predicates. Simply put, predicates are events or a condition that gives room for exploitable vulnerabilities in a system. This could be caused by failure to properly perform object type checks or other necessary input validations to ensure system safety. The examined functions are therefore transformed into predicate finite state machines (pFSMs) representing simple operations that could lead to the exploitation of such vulnerabilities. Finally a collection of all pFSMs derived were used to design the final finite state machine (FSM) model. Summarily, Chen identified 3 important observations needed to track down system vulnerabilities and foil attacks before the exploit is successful. These are:

- Software attacks are realized via a chain of elementary activities which can be broken at any point to foil the exploit.
- These exploiting activities involve the interaction and operations of several software objects.
- Source code analysis to determine these vulnerabilities will allow software security teams to specify conditions that must be met to provide security.

This method was very effective as further system vulnerabilities were uncovered during the research which was acknowledged by bugtraq, however, given the narrow spectrum of threats considered, the findings and recommendations cannot be generalized but carefully applied while designing threat models.

As a follow up, Sindre *et al* [10] proposed the adoption of misuse cases, a high-level behavioural model, to identify the possible security requirements an application would need to defend itself against impending attacks. Theses researchers drew inspiration from the idea of using use case models for functional specification during program design. However, misuse cases only specify the threats an adversary might pose to the system's functionalities and users but neither offers details about the attack sequences/steps that makes the exploit successful nor lends support for threat decomposition [20]. [8] in a bid to overcome this defect takes advantage of the UML sequence diagram's property of specifying objects interaction via messaging during program execution. Threat behaviors were viewed as misuse case scenarios and a sequence of object interactions documented by sequence diagrams. This method works by extracting threat execution paths at the design level. These paths are stored as signatures or threat traces that must never be performed during program execution. Wang argued that previous works only focused on confirming security violations through their threat models hence he took this a step further by ensuring that these threat models generate test cases for verification purposes thereby ensuring that the software implementation met the security requirements of the application. A major setback of his approach was its lack of real-time applicability in the industry though automated, it was heavily dependent on manual intervention by security experts to analyze the threat traces generated. It also involved a lot of reporting and trace generation which would be fed back into the system hence making the threat model grow out of manageable proportion thereby introducing more vulnerabilities and complexity. Finally, due to the method's dependence on misuse cases, unspecified threat paths within the resulting models might be exploited by seasoned attackers hence bypassing the security checks during program execution.

Subsequent to previous researches, [7] focused on resolving issues concerning test case generation from threat models using a mathematical modeling language called petri nets (Place/Transition Nets). As an abstract formal model of information flow, petri nets was used to model attack paths in web applications in the threat modeling exercise hence the name threat net was adopted. During the process of building the threat model with threat nets, similar security issues were classified in terms of the system's use cases and their associated threat categories. This approach heavily relied on the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) classification system for threat identification although the threat modeling industries and academic researchers have identified close to five or more different taxonomies for threat classification [21]. This introduces bias as the threat models from which test cases are generated may not represent all possible attacks. Given this defect, the method falls into the same category of allowing attackers exploit unspecified threat paths in the threat models.

Furthermore, security mutants [11] which were created during the experiment focused on a tiny spectrum of the vulnerability space such as those affecting C/C++

and web applications only therefore generated test cases cannot be used as a general representation of vulnerabilities but limited to the scope for which the research was executed [7].

Having examined previous threat modeling techniques to build defense and security into software applications, Marback *et al* approached Dianxiang's research from a whole different perspective using attack trees [11]. It is important to realize that the reason why all previous methods needed improvement was because the threat model on which all test cases were based had partial coverage hence security holes were not completely blocked. Marback's approach takes advantage of the unique feature of threat trees which starts with the attacker's goal as the root node, then branches out to all leaf nodes which represents possible attack sequences through which the main goal can be achieved. In essence, attack trees expose in detail all possible paths an attacker would traverse to achieve his objective of compromising the asset in view. This is a major advantage of threat trees over the previous techniques. Table 2 summarizes some of the threat modelling research activities executed till date.

HYBRID THREAT MODELING

Hybrid techniques involve the combination of two or more complementary threat modeling techniques to improve the security posture of the software asset being protected [22]. As identified in the threat modeling section, attack trees proved to be very effective in highlighting the goals and attack vectors liable to be exploited by hackers than all other techniques identified [11]. This is obviously responsible for its adoption in most of the hybrid threat modelling researches till date. By presenting every possible step an attacker traverses on a tree, pre-emptive mitigation steps can be taken to frustrate attacker's effort hence achieving deterrence.

Table-2. Threat modeling research activities.

Author (s), year	Technique	Test application	Focus/ attack vector
Marback *et al*, 2013	Attack Trees (ATs)	OS Commerce & Drupal CMS	Spoofing, Tampering & Denial of Service (DoS)
Dianxiang *et al*, 2012	Threat Nets	FileZilla, Magento	STRIDE
Mammar *et al*, 2012	Time Extended FSM / Timed Security rules	France Telecoms Travel Services	DoS
Wang *et al*, 2007	Sequence Diagrams	ATM Simulation	All Vulnerabilities
Sidre & Opdhal, 2005	Misuse Cases (MUCs)	Knowlegde Map App.	All Vulnerabilities
Chen *et al*, 2003	Predicate FSM & Source Code Analysis	Bugtraq Database	Stack, Heap & Integer Overflow, Input Validation and Format String vulnerabilities

Furthermore, several studies have revealed that misuse cases technically complements attack trees [3] as every single researcher used them in designing their hybrid threat model. This is obvious as the secure software development lifecycle dictates that security requirements must be stated first in the earliest part of the development process which is mostly done using misuse cases [14].

In order to take advantage of the strength of techniques (attack trees and misuse cases), [3, 23] performed experiments to compare these two techniques. The differences in Table 3 were established and more researchers such as Talukder [24] and Gondotra [25] in the same year came up with methods of exploiting their strengths in a hybrid technique.

Table-3. Differences between ATs & MUCs.

#	Misuse cases	Attack trees
1	Suitable for discovering threats in the early part of SDLC	Suitable for discovering threats in the later part of SDLC
2	Misuse cases are linked to users and high-level system functions through the use cases they build on. They are therefore particularly useful for generating new high-level threats and mitigations that are specific to the problem and solution domains for the new system.	Attack trees have the ability to systematically break down high-level threats into increasingly more detailed potential attack steps that may already be known. They are therefore particularly useful for identifying and tying together low-level threats corresponding generic security issues that apply to many types of systems.
3	A good starting point for security requirements engineering because they can be used for identifying an initial set of threats	Threats discovered can then be further refined using attack trees, combining the best of both techniques.
4	Misuse cases are solely based on pre-existing use case diagrams to provide a link between security requirements and the larger software development context	However, attack trees provide the necessary links via generating hierarchies of more detailed attack step sequences

Source: Comparing attack trees and misuse cases (Karpati *et al*, 2014; Opdahl and Sindre, 2009)

These set of differences gave insight into the possibility of combining both techniques to create better secured application. [24] proposed a tool called Suraksha, a term in Sanskrit which means safety and security, using

threat classifications schemes such as STRIDE and CI5A. The tool having benefited from both techniques does not represent the hybrid model with a unique notation to spell out weaknesses in the system. Also, all tasks were manual including report generation thereby loosing adoption due to poor usability and a high learning curve. Gandotra *et al* [25] combined both techniques creating a new set of notifications but faced difficulty as the model became complex within the hybrid technique. This was the same problem suffered by HARM [20] as six different techniques were combined including the previous two (ATs and MUCs) proposing a set of new modeling symbols that did away with almost all the well-known UML notations. HARM proved to be promising but the technique was marred by over specification of security requirements suffered through its application. HARM introduced many modeling guidelines that these techniques and symbols must represent while modeling the threat therefore leading to analysis paralysis [10]. Due to this setback, future researchers have gone back to the drawing board to use fewer techniques, as this research intends to do, but explore the possibility of reusable threat models which would nullify the paralysis issue. Hence, [22] designed a hybrid model that took advantage of the UML activity diagram rebranded as "security activity model". A repository of known and tested threat models called SHIELDS was also designed to store reusable threat models so as to nullify the issues caused by analysis paralysis. However, as a result of the difficulty in maintaining such repositories, this research work is yet to validate its results with the SHIELDS repository. However, work in progress shows this direction is a promising one hence the challenge of analysis paralysis due to over specification of security requirements still lingers. In summary, Table 4 highlights the remarkable research efforts in the field of hybrid threat modeling till date.

Table-4. Hybrid threat modeling research activities.

#	Researcher, year	Technique	Result/ outcome
1	Karpati *et al*, 2014	Attack Trees & Misuse Cases	Comparative Studies
2	Tondel *et al*, 2010	Security Activity Models, Attack Trees & Misuse Cases	SHIELDS
3	Karpati *et al*, 2010	Attack Sequence Description, Misuse Sequence Diagrams, Misuse Cases, MUC Maps, Attack Trees and AT Patterns	HARM – Hacker Attack Representation Method
4	Opdahl & Sindre, 2009	Attack Trees & Misuse Cases	Comparative Studies
5	Talukder *et al*, 2009	Attack Trees & Misuse Cases	Suraksha
6	Gandotra *et al*, 2009	Attack Trees & Misuse Cases	Hybrid Technique

DISCUSSIONS

Threat modeling is undoubtedly a very crucial exercise in the SSDL. It's unique capacity to identify and resolve vulnerabilities in the software at the earliest stages of development makes it a top priority for organizations running mission critical web applications. However, the threat-scape has changed immensely while attackers are more motivated and armed with sophisticated tools for doing serious damages. This is why researchers in both academics and the field of software security have made much emphasis on building security in at the earliest phases by adopting more advanced hybrid threat modelling techniques in order to enforce deterrence, cut development and maintenance cost while reducing the risk of damages in the event of a successful attack as much as possible. Having reviewed past literatures as seen in sections 4 and 5, it is important to summarize our finding as regards challenges facing the threat modelling field. These are:

- Software Security requirements suffer analysis paralysis due to over-specification while using hybrid threat modeling techniques
- Lack of reusable threat models and reliable threat repositories
- Incoherent set of acceptable modeling notations due to lack of coordination of researches from both the academic world and industry.

Given the objective of this paper which is to identify the current issues affecting threat modeling, this research intends to focus on the first challenge identified and further the studies on hybrid threat modeling by adopting the behavioral state machine (BSM) model alongside the previous two techniques (attack trees and misuse cases) in order to protect web applications from SQL injection vulnerabilities. State machines were chosen given its unique properties highlighted in [9, 26].

In order to adopt these three (3) techniques in the proposed hybrid threat model for preventing SQL injection in web applications, it is important to state and justify the order in which these techniques will be applied. Taking a cue from Table-3, misuse cases (dark ovals) in Figure-2 appear handy in discovering threats earlier than other techniques because it is directly related and connected to the high-level use case diagrams in the early phase of the SDLC.

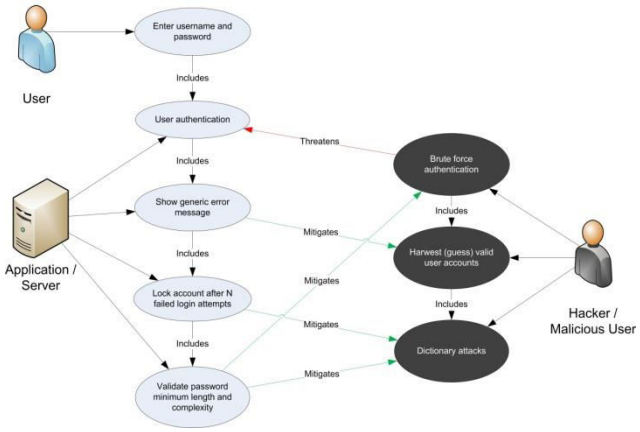

Figure-2. Misuse case representation of spoofing attack.
Source: Testing Guide Introduction. https://www.owasp.org

A general property of every application being developed is that it has a finite set of use cases throughout its lifetime till decommission. On the contrary, this does not mean misuse cases can be limited easily by the application's use cases as hackers can inject command into an application causing it to misbehave in ways never expected. This set of misbehaviors will be modeled as attack goals, which are the root nodes of attack trees the interacting objects must avoid. This reinforces the need to place attack trees as the next technique after misuse cases.

Take for instance a web application with a user authentication module which could be subjected to different kinds of spoofing attacks (misuse cases) especially SQL injection attacks as considered in this paper, then the relationship between this software objects can be depicted as shown in Figure-3. The result of connecting these two techniques will be an attack tree that specifies all attack vectors through which SQL injection can be achieved. An attack vector is simply a set of steps or a path an attacker traverses to achieve his goal of exploiting system vulnerability. In our case, the goal is to perform SQL injection.

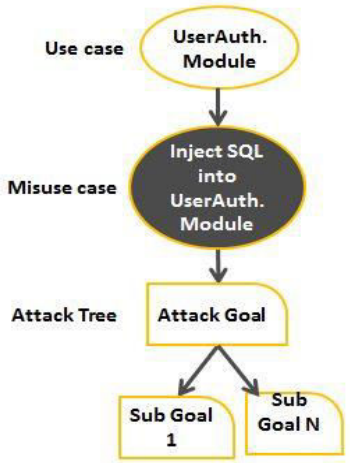

Figure-3. Connecting misuse cases to attack trees.

The algorithm below highlights the steps to be taken to prevent SQL injection Attacks in web applications

➔Start		
	1	*Convert all SQL injection types into misuse case scenarios*
	2	*Map each misuse case into Attack tree goals as the root node*
	3	*Transform all Attack Trees ➔ BSM*
	4	*Transform Software Asset (e.g. Authentication Module) ➔ BSM*
	5	*Merge Both BSMs from Step 3 and 4 to derive Software Asset with Exploitable Vulnerability (SAEV).*
	6	*Generate Security Test Cases from the SAEV model in step 5 and analyze result from test case execution to obtain Threat report*
	7	*Extract and recommend security requirements from Threat report*
	8	*Apply recommendations to affected software assets to resolve SQL injection vulnerabilities identified*
➔End		

Figure-4. Proposed hybrid threat modeling algorithm.

CONCLUSIONS

This paper presents threat modeling techniques and its hybrid approaches to testing the security of software applications especially web applications. It goes further to highlight the strength and weaknesses of these techniques identifying the reliable ones used in designing hybrid threat models namely; attack trees and misuse cases. The established differences of the two techniques shows they are complementary and careful adoption can improve the potency of the derived hybrid model. As highlighted in section 4, state machines have the unique capacity of breaking interacting software objects into simpler components thereby making it possible to control object specification, hence the additional technique to the previous two. Furthermore, related works were discussed showing the remarkable research activities carried out till date. The discussion section presented our findings, restated the objective of the paper work and showcased the proposed algorithm the researchers intend to implement in the future work

ACKNOWLEDGEMENTS

The authors would like to thank Universiti Tun Hussein Onn and the Office for Research, Innovation, Commercialization and Consultancy Management (ORICC) for funding this research under the Graduate Incentive Research Scheme (GIPS), VOT # U193.

REFERENCES

[1] B. Liu, L. Shi, Z. Cai, and M. Li, "Software vulnerability discovery techniques: A survey," in Multimedia Information Networking and Security

(MINES), 2012 Fourth International Conference on (pp. 152-156). IEEE. 2012, pp. 152-156.

[2] OWASP. (2014, 06-09-2014). Testing Guide Introduction. Available: https://www.owasp.org/index.php/Testing_Guide_Introduction

[3] P. Karpati, Y. Redda, A. L. Opdahl, and G. Sindre, "Comparing attack trees and misuse cases in an industrial setting," Information and Software Technology, vol. 56, pp. 294-308, 2014.

[4] A. Shostack, Threat modeling: Designing for security: John Wiley and Sons, 2014.

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

[6] F. Swideski and W. Snider, Threat Modeling: Microsoft Press, 2004.

[7] D. Xu, M. Tu, M. Sanford, L. Thomas, D. Woodraska, and W. Xu, "Automated security test generation with formal threat models," IEEE Transactions on Dependable and Secure Computing, vol. 9, pp. 526-540, 2012.

[8] L. Wang, E. Wong, and D. Xu, "A threat model driven approach for security testing," in Proceedings - ICSE 2007 Workshops: Third International Workshop on Software Engineering for Secure Systems, SESS'07, 2007.

[9] S. Chen, Z. Kalbarczyk, J. Xu, and R. K. Iyer, "A Data-Driven Finite State Machine Model for Analyzing Security Vulnerabilities," in Proceedings of the International Conference on Dependable Systems and Networks, 2003, pp. 605-614.

[10] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," Requirements Engineering, vol. 10, pp. 34-44, Jan 2005.

[11] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, "A threat model-based approach to security testing," Software-Practice & Experience, vol. 43, pp. 241-258, Feb 2013.

[12] D. Xu and K. E. Nygard, "Threat-driven modeling and verification of secure software using aspect-oriented Petri nets," IEEE Transactions on Software Engineering, vol. 32, pp. 265-278, 2006.

[13] USDHS. (2010, Software Security Testing. Software Assuarance Pocket Guide 3(0.7). Available: https://buildsecurityin.us-cert.gov/sites/default/files/software_security_testing.pdf

[14] C. Wysopal, L. Nelson, E. Dustin, and D. Dai Zovi, The Art of Software Security Testing: Identifying Software Security Flaws: Pearson Education, 2006.

[15] A. Avancini, "Security testing of web applications: A research plan," in Proceedings of the 34th International Conference on Software Engineering (pp. 1491-1494). IEEE Press, 2012, pp. 1491-1494.

[16] D. P. Mirembe and M. Muyeba, "Threat modeling revisited: Improving expressiveness of attack," in Proceedings - EMS 2008, European Modelling Symposium, 2nd UKSim European Symposium on Computer Modelling and Simulation, 2008, pp. 93-98.

[17] M. Paul, "Software security: Being secure in an insecure world," Retrieved September, vol. 1, p. 2009, 2008.

[18] A. Mammar, W. Mallouli, and A. Cavalli, "A systematic approach to integrate common timed security rules within a TEFSM-based system specification," Information and Software Technology, vol. 54, pp. 87-98, 1// 2012.

[19] Security Focus. (2014, 30/11/2014). bugtraq. Available: http://www.securityfocus.com/archive/1

[20] P. Karpati, G. Sindre, and A. L. Opdahl, "Towards a hacker attack representation method," in ICSOFT 2010 - Proceedings of the 5th International Conference on Software and Data Technologies, 2010, pp. 92-101.

[21] G. McGraw, Software security: building security in vol. 1: Addison-Wesley Professional, 2006.

[22] I. A. Tøndel, J. Jensen, and L. Røstad, "Combining misuse cases with attack trees and security activity models," in ARES 2010 - 5th International Conference on Availability, Reliability, and Security, 2010, pp. 438-445.

[23] A. L. Opdahl and G. Sindre, "Experimental comparison of attack trees and misuse cases for security threat identification," Information and Software Technology, vol. 51, pp. 916-932, 2009.

[24] A. K. Talukder, V. K. Maurya, S. Babu G, J. Ebenezer, M. Sekhar V, K. P. Jevitha, *et al.*, "Security-aware software development life Cycle (SaSDLC) - Processes and tools," in 2009 IFIP International Conference on Wireless and Optical Communications Networks, WOCN 2009, 2009.

[25] V. Gandotra, A. Singhal, and P. Bedi, "Identifying security requirements hybrid technique," in 4th International Conference on Software Engineering Advances, ICSEA 2009, Includes SEDES 2009: Simposio para Estudantes de Doutoramento em Engenharia de Software, 2009, pp. 407-412.

[26] O. El Ariss, J. Wu, and D. Xu, "Towards an enhanced design level security integrating attack trees with statecharts," in Proceedings - 2011 5th International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 2011, pp. 1-10.