J. Basic. Appl. Sci. Res., 3(4)642-647, 2013
© 2013, TextRoad Publication

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

A Review of Approaches to Model Security into Software Systems

Shafiq Hussain[1], Ghulam Rasool[2], Muhammad Atef[2], Abdul Karim Shahid[2]

[1]Department of Computing, Engineering and Technology, University of Sunderland, Sunderland, United Kingdom
[2]Department of Computer Science, COMSATS Institute of IT, Lahore, Pakistan

ABSTRACT

Software security has a huge impact on almost all areas ranging from banking systems to critical systems. The rapid expansion of internet and distributed systems has forced developers, designers, engineers and manager to consider software security as an essential activity for their systems. Software security does not depend on the external measures such as firewalls but also on the internal security of software applications. Internal security of software systems is a major concern of current software systems. A number of methodologies have been developed over the time to address the issues of software security. In this paper, a survey of these methodologies has been presented. This paper surveys the methodologies only used for the internal security of software systems. The methodologies used for external security of software systems are not in the scope of this paper. This survey has focussed on four parameters of the methodologies: model driven methodologies, methodologies having automatic tool support, methodologies having no tool support and methodologies based on formal methods. A critical analysis of the methodologies is also presented. Future research directions are also discussed based on the critical analysis to develop a more secure methodology for software systems.

KEYWORDS: Security, Development Methodology, Formal Methods, SDL, TAM, Tropos, SQUARE.

1. INTRODUCTION

The rapid growth of distributed systems and internet software security has jumped from external protections to highly secure design of software systems. The software systems are more vulnerable to security attacks now as compared to past when internet was not so advanced [60]. Attackers have become advanced and applying sophisticated techniques to breach the security of software system and gaining access to devices they are controlling [11]. The increasing dependence of devices on computer for automation and accuracy and weak links of network mechanisms are another reason of breaches. Therefore, more consideration is required to address security issues of software systems [62]. In the context of current systems where information, processing and infrastructure are distributed the security issues of software systems becomes more critical [20][45]. In the past main focus was on capturing functional properties of software systems and very little attention was given to non-functional behaviours such as security [59]. Whole concentration was made to protect software systems from outside environment and controls such as firewalls were installed to protect them [19]. There was no particular methodology in practice and whole dependence was on best practices. It was soon realized that external protections in the form of firewalls, network level control and operating systems level controls were not much enough to secure software [43]. Software security issues should be addressed at the design level of systems[32]. To ensure security into the software systems, software designs must be analysed, verified and validated using mathematical techniques such as formal methods, theorem provers and model checkers [1] [29]. The rest of the paper is organised as follows: Section 2 describes a survey of model driven methodologies. Section 3 surveys methodologies having automatic tool support. Section 4 presents a survey of methodologies having no automatic tool support. Section 5 introduces a survey on methodologies based on formal methods.

2. Model Driven Methodologies

In Model Driven Methodologies, models are used as the basis for the development of software systems. The models of software systems are developed starting from specification to design. These models are platform independent. The resulting models can be implemented on any machine, any operating system and any programming language.

2.1 Model Driven Architecture (MDA). Model Driven Architecture (MDA) is a methodology based the development of system models and then all the other phases of systems are derived from the previous models. Model Driven Architecture methodology helps to specify software systems without taking care of platforms for their implementation. In Model Driven Architecture methodology, basic component for developing systems are models and are used for the production of specification, construction of design, testing of properties, deployment of systems, operations of the system and maintenance. The key features of this methodology are portability, interoperability and reusability[46] [40] [41][4].

2.2 Model Driven Security. Model Driven Security uses Secure UML which is an extension to UML and is based on Model Driven Architecture. Secure UML is used to generate the system architectures from the system models. Then security mechanisms are generated from these models [5] and [34]. Secure UML defines nine stereotypes to annotate a class diagram with role-based access control information. Secure UML uses OCL (Object Constraint Language) to specify constraints for assets, actions and

Corresponding Author: Shafiq Hussain, Department of Computing, Engineering and Technology, University of Sunderland, Sunderland, United Kingdom. Tel: +447466638765 Email: shafiq.hussain@research.sunderland.ac.uk

permissions. This methodology is lacking automatic transformation from models to code [55].SecureUML is a generic security policy language and it focuses on modelling access control to protected resources.

3. Methodologies with Automatic Tool Support

In these methodologies there is an automatic tool support for the development of software systems. The automatic tool may be an editor, an analyser or some other tool facilitating the process.

3.1 UMLSec. UMLSec, an extension to UML is a methodology to specify security requirements. It uses stereotypes, tags, constraints, class diagrams, state charts, activity diagrams, sequence diagrams and deployment diagrams to model security into components of the systems. This approach recommends multilevel security and mandatory access control. UMLSec consists of 21 stereotypes used to specify security requirements such as non-repudiation, role-base access, fair exchange, secure communication link, confidentiality, integrity, authenticity etc. UMLsec has been integrated with CASE tools that support automated analysis routines in order to verify the models developed with UMLsec against security requirements by utilizing the constraints associated with the UMLsec stereotypes. Models in UMLsec can be verified by using model checking[31][6][47].

3.2 Abstract State machine language (AsmL) Abstract State machine language (AsmL) is a software specification language.Attack scenarios are specified in Asml and later on Snort rules are derived from them. Then these rules are used in Snort, an Intrusion Detection System (IDS). Attacks with multiple steps can be specified in AsmL[26].

3.3 AsmLSec. AsmLSec, an extension to AsmL uses states, events and transitions to represent attacks. Each transition has a source state, destination state, a set of conditions to fire a transition and actions to be performed in case the transition is fired. Attack scenarios modelled in AsmLSec can be translated to AsmL which are used in IDS[50].

3.4 Snort. Snort, an intrusion detection system (IDS) used to detect an attack over a network by using attack scenarios as rules. It specifies the actions to be taken if a rule is matched with a network packet, the source and destination IP address and ports, the protocol of the observed network and direction of the network packet[51].

3.5 STATL. State/Transition based Attack description Language (STATL)is based on finite state machine. It used state and transition to specify an attack. Transitions have events to fire a transition and actions. STATL specifications can have executable code and a development environment along with visualization facilities of attack scenarios. The most important features of STATL are simplicity, expressiveness, rigor, extensibility, excitability, translatability, portability and heterogeneity. The STATL is supported by a graphical development editor to visualise the models. Attack scenarios in STATL are used by Intrusion Detection Systems for the detection of intrusions in the networks. A toolset has been developed to implement STATL based intrusion detection systems. Attack scenarios defined in STATL are domain independent and can be used in different environments such as UNIX, Linux, Windows, and Solaris[15].

3.6 Secure Tropos. Secure Tropos is an Agent-Oriented Software Development Methodology covering all the stages of SDLC from requirements to implementation. It is based on the idea of incremental development. Secure Tropos is a four step methodology in which actors, goals, soft goals, their dependencies and their security requirements are identified as first step. Then further goal and sub goals are identified by in depth analysis of first step activities. New security constraints are identified based on new goals. At the final stage, tasks are defined to achieve sub goals. These constraints are the security requirements. SecTro is a tool to support development with Secure Tropos[42][25] [8].

3.7 Security Quality Requirements Engineering (SQUARE).SQUARE is a methodology for secure software system development. It is a 9 step process consisting of: agree on definitions, identify security goals, develop supporting artefacts, perform risk assessment, select elicitation techniques, elicit security requirements, categorize requirements and prioritize requirements. It has a tool support called SQUARE Tool to automate all of the above nine steps [37].

3.8 Security Requirement Engineering Process. Security Requirement Engineering Process (SREP) is a process to capture security requirements. It is based on Common Criteria and has support of an automatic tool, SREP Tool. The basic theme of this process is to reuse the security requirements and add reusable components to Security Resources Repository (SRR). SREP process consists of 9 phases: agreement on security definitions, identification of critical assets, identification of security objectives, and identification of threats using misuse cases or attack trees, assessment of asset risk, extraction of security requirements, categorization and prioritization of security requirements, requirements inspection and addition of reusable components to SRR[39].

3.9 Microsoft Secure Development Lifecycle (SDL). Microsoft Secure Development Lifecycle (SDL) is a complete development process based on the principles of secure by development, secure by default, secure by deployment and communication. In SDL development process, user requirements are identified, high impact security components are highlighted, design techniques are finalized, access points are identified, threat modelling and risk analysis is performed and security requirements are identified. In addition, completion criteria, testing, code reviews, static code analysis using tools, security code reviews and security testing to identify security objectives are finalized. It recommends the use of mitigation techniques in the design phase and secure coding practices in implementation phase [33].

3.10 Comprehensive, Lightweight Application Security Process (CLASP). Comprehensive, Lightweight Application Security Process is a security requirements engineering process.in this process first of all, resources, assets and roles are identified. Secondly, resources and assets are categorized into classes according to the level of required security required. Interactions for resources are identified. Based on the interactions of assets, security requirements are identified including authorization, authentication, integrity, confidentiality, availability and accountability. CLASP has a tool support. CLASP is also secure software development process consisting of activities according to roles defined during development. CLASP also suggests the assignment of a security expert from the beginning of development. For the requirement specification phase, CLASP advocates performing risk analysis and threat modelling. An attacker profile should also be developed based on the potential attacker and their resources. Moreover, the attack surface and security features that need to be implemented should be identified. According to CLASP, risk analysis and threat modelling should be performed again during the design phase. CLASP proposes to annotate class diagrams with security information. In the security assurance phase, CLASP recommends performing security code reviews, security code scanning, and security testing. CLASP also provides a list of common vulnerabilities with comprehensive information about how and when they can be introduced during development and how to avoid them [59] [14].

4. Methodologies without Tool Support

In these methodologies there is no tool support helping to automate the process of development. These methodologies are just frameworks, processes or guidelines.

4.1 UMLintr. UMLintr is an extension to UML in which stereotypes and tags are used to specify attacks using case diagrams, class diagrams, state charts and package diagrams. UMLintr (UML for intrusion specification) is used to specify intrusion scenarios. It eliminates the need of a separate attack specification language. Authors described a framework for intrusion identification, intrusion specification and generation of intrusion signatures to be used in intrusion detection systems. But the authors did not describe any automated tool[30].

4.2 Misuse cases. Misuse cases are a special type of UML use case that describes the misbehaviour of the software. A misuse case is started by a misact or which in turn along with misuse case can be used to generate more use cases to eliminate the threats posed by misuse cases. First use cases and actors are specified then misuse cases and mis-actors are specified. After that the relationships between use cases and misuse cases are identified. Then new use cases are specified. These new use cases form the high level security requirements of the software[54][7][61].

4.3 Abuse cases. Abuse cases are another way for the specification of undesirable behaviour. Here abuse case model is obtained by using UML use case diagrams and there is no notational difference between use case diagram and abuse case model. All harmful interactions are identified by using actors and abuse cases. All the approaches to perform a particular attack (e.g.sql injection) are specified in a tree structure and hence identification of all possible approaches for all possible attacks. A textual detail of all actors must also be included[35].

4.4Haley Framework. Haley framework unifies the requirements engineering and security engineering by getting functional requirements from requirements engineering and security requirements from security engineering along with assets, threats to those assets and goals to protect those assets[27].

4.5 McGraw's Secure Development Life Cycle Process. McGraw's Secure Development Life Cycle Process recommends the addition of some secure software development activities to conventional SDLC during different development phases. These activities include risk analysis, risk-based security testing, static analysis and penetration testing guided by abuse cases and security requirements[36].

4.6 Appropriate and Effective Guidance for Information Security (AEGIS). Appropriate and Effective Guidance for Information Security (AEGIS) is an SSDLC process based on spiral model and recommends the identification of assets and performing risk analysis. It consists of four session process. During first and second phase abuse cases are used to identify assets and their relationships along with requirements about confidentiality, integrity and availability. In third session, risk, vulnerabilities and threats are identified. In fourth session security requirements are identified. The risk analysis methods includes determination of vulnerabilities, determination of cost and likelihood of an attack in the deployed environment, selection of security requirements based on security expert's advice, cost-benefit assessment of the selected security requirements, comparison between the cost and likelihood of each attack against the cost of security requirement and selection of security requirements based on cost effectiveness. Modelling language is UML[18].

4.7 Secure Software Development Model (SSDM). Secure Software Development Model (SSDM) is a waterfall based model in which all potential attacks are identified using threat modelling in the requirement engineering phase. Based on threats, a security policy will be designed to be followed through the whole process and assurance will be achieved by penetration testing. It is just a methodology with no tool support[57].

4.8 Aprville and Pourzandi's Secure Software Development Life Cycle Process. Aprville and Pourzandi's Secure Software Development Life Cycle Process is based on the identification of security objectives, identification of security requirements by threat modelling, prioritization of security requirements, representation of design decisions by UMLSec, choosing proper secure programming language, usage of verified algorithms for cryptography, code reviews, static vulnerabilities code scanners, ad-hoc unit and system security testing and fuzz testing[49].

4.9 Secure Software Development Process Model (S2D-ProM). Secure Software Development Process Model specifies multiple possible strategies to progress from one step to another step. The process proposes to conduct risk analysis during requirement specification, design, and implementation phases. Risk analysis, according to S2D-ProM, can be performed in different ways for each development phase. For example, error checklists or personal experience can be used in the requirement specification phase, model checking or design reviews can be used in the design phase, and testing or code reviews can be used in the implementation phase. S2D-ProM also provides multiple options while advancing from one development phase to another. For example, security standards or personal experience can be used to develop requirement specifications, a security modelling language or security patterns can be used to construct design from requirements, and secure coding rules or personal experiences can be used to develop source code from design. S2D-ProM does not specify whether only one strategy should be used while moving from one phase to another or multiple strategies can be used at the same time [16].

4.10 Team Software Process for Secure Software Development (TSP-Secure).TSP-Secure process consists of three components: planning for security, managing quality and security throughout the development life cycle, and educating developers about security related aspects. During the planning phase the team identifies security goals and produces a detailed plan to guide development. Development activities in the plan may include identifying security risks, eliciting security requirements, secure design, code reviews, unit testing, fuzz testing, and static code analysis. The team may choose any SSD activity as deemed necessary. The security manager manages all the security related activities in the system development [12].

5. Methodologies based on Formal Methods.
These methodologies make use of formal methods for specification of requirements, design of models, verification of models and validation of properties. Formal methods are tools and techniques based on mathematical logic and are used in all phases of software system development life cycle.

5.1 Software Security Assessment Instrument (SSAI). Software Security Assessment Instrument is methodology based on resources and tools to help in developing secure software. The first resource of SSAI provides is an online vulnerability database containing information about various vulnerabilities and their exploits and mitigations. The second SSAI resource is a security checklist that can be used to guide development process. Authors present details on how to develop such a checklist and potential items that can be included. The third resource is a list of publicly available static code scanning tools. SSAI also provides flexible modelling framework (FMF) which is a modelling tool. The FMF can be used to develop models and verify them for desired properties using model checking. By using this methodology, state explosion problems can be delayed Lastly, SSAI provides a property-based testing tool (PBT) which uses the security properties specified in the security checklist or FMF as a basis to test the software[21].

5.2 Correctness-by-Construction. Correctness-by-Construction methodology makes use of formal methods at all the phases of development life cycle of software. In this approach, software systems are specified by using formal specification languages and the resulting models are verified and validated by using model checkers and theorem provers. Only security critical components of software systems are developed using formal methods by using this methodology [28][2][49].

6. DISCUSSIONS

This is not an exhaustive survey of all the techniques and methodologies. In this survey the focus is on more relevant and practical methodologies. Every methodology has merits and demerits. The methodologies presented in section 2 are model driven methodologies. These methodologies are suitable for those systems only where implementation platform is irrelevant. These methodologies are very good for systems where systems are distributed and run on different platforms. These systems require integration at different points in their deployment. For specific systems which require particular platform, these methodologies are not good choice. The methodologies described in section 3 are much better methodologies for designing secure software systems. These methodologies also have tool support to automate the development process. But these methodologies are lacking rigorous analysis. These are based on informal and semi-formal techniques. Requirements cannot be captured precisely in these methodologies due to ambiguities present in these techniques. Therefore, resulting systems are not consistent and hence leads to security breaches. The methodologies presented in section4 are processes, frameworks or guidelines. These are good for guiding designers, engineers and managers. These methodologies describe guidelines for developing secure software systems. There is no tool support for these methodologies. Also, these methodologies do not have rigorous analysis mechanisms to improve the systems at design level. These methodologies do not have support for mathematical notations for describing the systems. Therefore, systems developed by using these methodologies may always be inconsistent. The methodologies described in section 5 are based on formal methods. Formal methods are mathematics based approaches for the specification, design, analysis, verification and validation of software systems. Since, these methodologies are based on mathematics, software systems can be specified, designed and analyzed rigorously. Hence, resulting systems are consistent, complete and unambiguous. These systems can also be verified automatically at the design level before implementation. Hence, reduces the cost of exhaustive testing. The systems can be validated by model checkers and theorem provers. But formal methods have very little applications in computer security as we saw in this survey. Formal methods have successful applications in the area of critical systems. Their use is very limited in industry due to lack of experts in this area.

7. Future Research Directions

A number of secure software development methodologies has been described in this survey. Formal methods are very good for the specification, design and analysis of software systems due to their rigorousness and analysis facilities. Formal methods should be used to software development process. But formal methods are very heavy and a lot of expertise is required to use them particularly to generate formal proofs of systems properties. Therefore, lightweight application of formal methods is highly recommended. A new methodology needs to be developed based on lightweight application of formal methods for developing secure software applications. Tool should be developed to automatically transform user requirements into formal specifications. A new methodology based on integration of formal methods should be developed. Because some formal methods are good for specification, some are good for design and some are good for verification and validation of properties. Therefore, a more complete and rigorous methodology based on integration of formal methods needs to be investigated.

8. Conclusion

A survey of secure development methodologies for software systems has been presented. The survey focussed on four types of methodologies consisting of model driven architecture, methodologies based on automatic tool support, methodologies without automatic tool support and methodologies based formal methods. A critical analysis of these methodologies is presented which can guide research to select any particular methodology according to his/her requirements. Future research directions are also highlighted.

REFERENCES

1. Ali, G., Khan, S.A., Ahmad, F., Zafar, N.A., 2012. Visualized and Abstract Formal Modelling towards the Multi-Agent Systems,*J. Basic. Appl. Sci. Res* 2(8), 8272-8284.
2. Amey, P., 2002. Correctness by Construction: Better can also be Cheaper. *CrossTalk: the Journal of Defense Software Engineering*, 2, pp.24–28.
3. Apvrille, A. &Pourzandi, M., 2005. Secure software development by example. *Security & Privacy, IEEE*, 3(4), pp.10–17.
4. Aytaç, S., 2006. MODEL DRIVEN ARCHITECTURE.
5. Basin, D., Doser, J. &Lodderstedt, T., 2003. Model driven security for process-oriented systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*. p. 109.
6. Best, B., Jurjens, J. &Nuseibeh, B., 2007. Model-based security engineering of distributed information systems using UMLsec. In *Proceedings of the 29th international conference on Software Engineering*. pp. 581–590.
7. Braz, F.A., Fernandez, E.B. &VanHilst, M., 2008. Eliciting security requirements through misuse activities. In *Database and Expert Systems Application, 2008. DEXA'08. 19th International Conference on*. pp. 328–333.
8. Bresciani, P. et al., 2004. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3), 203–236.
9. Casteele, S.V., 2005. Threat modeling for web application using STRIDE model.
10. Chen, Y., Boehm, B. & Sheppard, L., 2007. Value Driven Security Threat Modeling Based on Attack Path Analysis. In *HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*. p. 4698.
11. Choo, K.K., Smith, R.G. &McCusker, R., 2007. The future of technology-enabled crime in Australia. *Trends & issues in crime and criminal justice*, 341, 341–360.
12. Davis, N., 2005. *Secure Software Development Life Cycle Processes: A Technology Scouting Report,* Citeseer.
13. De Cock, D. et al., 2004. Threat modelling for security tokens in web applications. In *Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004)*.
14. De Win, B. et al., 2009. On the secure software development process: CLASP, SDL and Touchpoints compared. *Information and Software Technology*, 51(7), 1152–1171.
15. Eckmann, S.T., Vigna, G. & Kemmerer, R.A., 2002. STATL: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1), 71–103.
16. Essafi, M., Labed, L. & Ben Ghezala, H., 2007. S2D-ProM: A Strategy Oriented Process Model for Secure Software Development. In *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*. pp. 24–30.
17. Firesmith, D., 2004. Specifying reusable security requirements. *Journal of Object Technology*, 3(1), 61–75.
18. Flechais, I., Sasse, M.A. &Hailes, S., 2003. Bringing security home: a process for developing secure and usable systems. In *Proceedings of the 2003 workshop on New security paradigms*. p. 57.
19. George, V. & Vaughn, R., 2003. Application of lightweight formal methods in Requirement Engineering1. *CrossTALK-The Journal of Defense Software Engineering (Jan 2003)*, 16(1), pp.30–32.
20. Gilaninia, S., Mousavian, S.J., Taheri, O., Nikzad, H., Mousavi, H., ZadbagherSeighalani, F., 2012. Information Security Management on performance of Information Systems Management. *J. Basic. Appl. Sci. Res* 2(3), 2582–2588.
21. Gilliam, D. et al., 2003. Addressing software security and mitigations in the life cycle. In *Proceedings of the 28th Annual NASA Goddard IEEE Software Engineering Workshop (SEW)*. pp. 201–206.
22. Gilliam, D.P. et al., 2001. Development of a software security assessment instrument to reduce software security risk. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001. WET ICE 2001. Proceedings. Tenth IEEE International Workshops on*. pp. 144–149.
23. Gilliam, D.P. et al., 2003. Software security checklist for the software life cycle. *Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings*, 243–248.
24. Gilliam, D.P., Kelly, J.C. & Bishop, M., Reducing software security risk through an integrated approach. In *Proc. of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June, 2000), Gaithersburg, MD*. pp. 141–146.
25. Giorgini, P., Mouratidis, H. &Zannone, N., 2006. Modelling Security and Trust with Secure Tropos. *Integrating Security and Software Engineering: Advances and Future Vision*, 160–189.
26. Graves, M. &Zulkernine, M., 2006. Bridging the gap: software specification meets intrusion detector. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. PST ’06. New York, NY, USA: ACM, pp. 1–31.
27. Haley, C.B. et al., 2008. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1), 133–153.
28. Hall, A. & Chapman, R., 2002. Correctness by construction: Developing a commercial secure system. *IEEE software*, pp.18–25.
29. Hussain, S., Erwin, H. & Dunne, P., 2011. Threat modeling using formal methods: A new approach to develop secure web applications. In *Emerging Technologies (ICET), 2011 7th International Conference on*. pp. 1–5.
30. Hussein, M. &Zulkernine, M., 2006. UMLintr: a UML profile for specifying intrusions. In *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*. p. 8.
31. Jürjens, J., 2002. UMLsec: Extending UML for secure systems development. *«UML» 2002—The Unified Modeling Language*, 1–9.

32. Kim, J., Kim, M. & Park, S., 2006. Goal and scenario based domain requirements analysis environment. *The Journal of Systems & Software*, 79(7), 926–938.
33. Lipner, S. & Howard, M., 2004. The trustworthy computing security development lifecycle. In *20th Annual Computer Security Applications Conference (ACSAC 2004)*. pp. 2–13.
34. Lodderstedt, T., Basin, D. &Doser, J., 2002. SecureUML: A UML-based modeling language for model-driven security. *«UML» 2002—The Unified Modeling Language*, 426–441.
35. McDermott, J. & Fox, C., 1999. Using abuse case models for security requirements analysis. In *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*. pp. 55–64.
36. McGraw, G., 2006. *Software security: building security in*, Addison-Wesley Professional.
37. Mead, N.R. &Stehney, T., 2005. Security quality requirements engineering (SQUARE) methodology. In *Proceedings of the 2005 workshop on Software engineering for secure systems—building trustworthy applications*. p. 7.
38. Mellado, D. et al., 2010. A systematic review of security requirements engineering. *Computer Standards & Interfaces*.
39. Mellado, D., Fernández-Medina, E. &Piattini, M., 2007. A common criteria based security requirements engineering process for the development of secure information systems. *Computer Standards & Interfaces*, 29(2), 244-253.
40. Miller, J., Mukerji, J. & others, 2001. Model driven architecture (mda). *Object Management Group, Draft Specification ormsc/2001-07-01*.
41. Miller, J., Mukerji, J. & others, 2003. MDA Guide Version 1.0. 1. *Object Management Group*, 234.
42. Mouratidis, H. &Giorgini, P., 2007. Secure tropos: A security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2), 285–309.
43. Muniraman, C. &Damodaran, M., 2007. A practical approach to include security in software development. *Issues in Information Systems*, 2(2), 193–199.
44. Myagmar, S., Lee, A.J. &Yurcik, W., 2005. Threat modeling as a basis for security requirements. In *Symposium on Requirements Engineering for Information Security (SREIS)*.
45. Nikakhtar, N., Jianzheng, Y., 2012. Risk Management Model of Electronic Commerce by the help of Decision support system. *J. Basic. Appl. Sci. Res* 2(1) 118-124.
46. Poole, J.D., 2001. Model-driven architecture: Vision, standards and emergingtechnologies. In *Workshop on Metamodeling and Adaptive Object Models, ECOOP*.
47. Popp, G. et al., 2003. Security-critical system development with extended use cases. In *Software Engineering Conference, 2003. Tenth Asia-Pacific*. pp. 478–487.
48. Potter, B., 2009. Microsoft SDL Threat Modelling Tool. *Network Security*, 2009(1), 15–18.
49. Pourzandi, M., 2005. Secure Software Development by Example. *IEEE Security & Privacy Magazine*, 3(4), 10–17.
50. Raihan, M. &Zulkernine, M., 2007. AsmLSec: an extension of abstract state machine language for attack scenario specification. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*. pp. 775–782.
51. Roesch, M., 1999. Snort-lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*. pp. 229–238.
52. Saini, V., Duan, Q. &Paruchuri, V., 2008. Threat modeling using attack trees. *J. Comput. Small Coll.*, 23(4), 124-131.
53. Shostack, A., 2008. Experiences Threat Modeling at Microsoft. In *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*.
54. Sindre, G. &Opdahl, A.L., 2005. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1), 34–44.
55. Smith, S., Beaulieu, A. & Phillips, W.G., 2009. Modeling Security Protocols Using UML 2.
56. Sodiya, A.S. et al., 2007. Threat Modeling Using Fuzzy Logic Paradigm. *Information and Beyond: Part I*, 4, 53.
57. Sodiya, A.S., Onashoga, S. &Ajayi, O.B., 2006. Towards building secure software systems. *Issues in Informing Science & Information Technology*, 3, 635–646.
58. Verdon, D. & McGraw, G., 2004. Risk analysis in software design. *IEEE Security & Privacy*, 79–84.
59. Viega, J., 2005. Building security requirements with CLASP. In *Proceedings of the 2005 workshop on Software engineering for secure systems—building trustworthy applications*. p. 7.
60. Walton, J.P., 2002. Developing an enterprise information security policy. In *Proceedings of the 30th annual ACM SIGUCCS conference on User services*. pp. 153–156.
61. Whittle, J., Wijesekera, D. &Hartong, M., 2008. Executable misuse cases for modeling security concerns. In *Proceedings of the 30th international conference on Software engineering*. pp. 121–130.
62. Zulkernine, M. &Ahamed, S., 2006. Software security engineering: Toward unifying software engineering and security engineering. *Enterprise Information Systems Assurance and Systems Security: Managerial and Technical Issues, M. Warkentin& R. Vaughn*, 215–232.