EPiC Series in Computing
Volume 58, 2019, Pages 180–190
Proceedings of 34th International Conference on Computers and Their Applications

Software Engineering Principles and Security Vulnerabilities

Aakanksha Rastogi and Kendall E. Nygard
Department of Computer Science
North Dakota State University, Fargo, ND, USA.
{aakanksha.rastogi,kendall.nygard@ndsu.edu}

Abstract

Software Engineering principles have connections with design science, including cybersecurity concerns pertaining to vulnerabilities, trust and reputation. The work of this paper surveys, identifies, establishes and explores these connections. Identification and addressing of security issues and concerns during the early phases of software development life cycle, especially during the requirements analysis and design phases; and importance of inclusion of security requirements have also been illustrated. In addition to that, effective and efficient strategies and techniques to prevent, mitigate and remediate security vulnerabilities by the application of the principles of trust modelling and design science research methodology have also been presented.

1 Introduction

The continuous evolution of software systems brings associated risks, vulnerabilities, and threats to their security. Vulnerabilities make software systems open to exploitation and attacks. Common types of attacks experienced by enterprises and consumers are caused by virus, worms, phishing, spams, and similar malware. According to Peissens, “A vulnerability is any aspect of a computer system that allows for breaches in its security policy” [44]. Demchenko et. al defines vulnerability as “a flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy” [13]. Vulnerabilities caused by software bugs, defects, anomalies, or software features are referred to as software vulnerabilities. A software vulnerability is a security weakness, flaw or a glitch found in a software or an operating system that leads to security concerns. Software vulnerabilities constitute a majority of security problems and are prone to exploitation for purposeful disasters. Some of the common flaws or vulnerabilities include weak or flawed authentication, buffer overflow, configuration problems, CRLF injection, cryptographic error (poor design or implementation), CrossSite Request Forgery (CSRF), insecure default configuration (e.g. passwords or permissions), design problems, DoS flood [10], DoS caused by malformed input [10], DoS due to improper release of services [10], Double free vulnerability, Eval injection, format string vulnerability, integer overflow, symbolic link following, memory leak, unescaped metacharacters or other unquoted 'special' characters, default or hard-coded password, PHP remote file inclusion, bad privilege assignment, unprotected/unauthenticated privileged process/action, untrusted search path vulnerability, spoofing attacks [10], SQL injection vulnerability [10], Cross-site scripting [10], SSL related vulnerabilities (HeartBleed and Poodle) [41], DOM-based XSS [10] , stored XSS [41], Transport Layer Security (TLS) and Secure Socket Layer (SSL) related vulnerabilities [41]. Vulnerabilities can creep in at any stage of software development and can disrupt the viability of a software product. Since security is a non-functional requirement, it is often an afterthought and given less importance than functional requirements throughout the stages of the software development lifecycle (SDLC). The effect being that the product-to-be-developed can obliviously become prone to security vulnerabilities. Each software vulnerability is distinct and hence requires different approaches to detect, assess, and mitigate the problem. Pescatore categorized software vulnerabilities into software defects (comprising of subcategories – design flaws and coding flaws) and configuration errors (comprising of subcategories – dangerous/unnecessary services and access administration errors) [43]. Since, each software vulnerability is one of a kind, they cannot be treated equally and hence, require different ways of assessment and mitigation approaches. To achieve higher levels of security while reducing costs, it is important to orient vulnerability assessment and mitigation approaches towards the more likely causes of defect [43]. Some of the ways to discover and identify software vulnerabilities include smart fuzzing [8] [5] [40] [32] [25], statically scanning java code [60], using vulnerability discovery model (VDM) [26], attack injection [3] [58] and code clone verification [33].

Security vulnerabilities in software systems are detrimental to the reputation of a system and negatively impacts user trust in that system. A hint of security issues in a software system will drive potential users away from its usage and adoption. Hence it is of high importance that security vulnerabilities are identified early in the requirements engineering phase and preventative measures and approaches are explored, acknowledged, and defined in the design phase to control these vulnerabilities. This paper identifies and explores software security vulnerabilities and explains their importance in the early phases of software development. An illustration of parallels between security approaches and practices in software engineering and design science is also presented. The rest of the paper is organized as follows. Section II identifies, categorizes, and describes cybersecurity vulnerabilities in the requirements engineering phase of the software development life cycle and emphasizes on the establishment of security requirements during the analysis phase. Section III identifies, categorizes, and describes cybersecurity vulnerabilities in the design phase of the software development life cycle. Section IV describes the impact of cybersecurity vulnerabilities in software systems on the trust and reputation of the system and studies benefits of trust modeling. Section V discovers and presents a parallel connection between security in software engineering and in design science in terms of security patterns. Section VI wraps up the paper with conclusion.

2 Requirements and Points of Vulnerability

Security being a non-functional requirement, it is often given less importance over other functional requirements during the requirements elicitation phase. This makes development and usage of software applications vulnerable to cybersecurity attacks and affects credibility. To ensure the development of a secure software product, it is important that security aspects are acknowledged and addressed from the beginning of the software development life cycle, which is at the requirements gathering and elicitation phase. A common problem with the specification of security requirements is their tendency to be replaced by security specific architectural constraints. These constraints may restrict the security team from using the most apt security mechanisms for the purposes of achieving true underlying security requirements [52]. Moreover, the amount of attention given to security requirements is often overpowered by the attention given to the functional requirements while only considering the user's point of view. Prioritizing convenience and ease of use over security requirements often leads designers and developers to only work on generic set of security requirements that have general mechanisms such as authorization, password protection, authentication, access control, encryption, firewalls, signatures, and virus detection tools [52].

During the analysis or requirements specification phase of software development, the developers should perform a high-level abstraction of risk analysis and formulate suitable security requirements. This practice can help avoid introduction of common security flaws in this stage [44]. A security vulnerability taxonomy broadly classifies these security flaws into three categories viz. No Risk Analysis, Biased Risk Analysis and Unanticipated Risks [44]. No Risk Analysis denotes development of software without any security issues in mind. During an analysis phase, when the software engineer skips past the acknowledgment of potential risks and a suitable security policy for the software, it ascertains vulnerabilities in the software product [44]. Biased Risk Analysis denotes risk analysis being completed by only one of the stakeholders which only accounts for the security requirements formulated per the position of that particular stakeholder, often ignoring requirements of other stakeholders [44]. Since security is based on mutual distrust, risk analysis process should include risk of all parties [44]. Unanticipated risks denote failure of a developer to recognize a risk and a failure to include a corresponding security requirement for the same [44].

It takes a combination of software development, subjective judgment of security professionals, and business expertise to detect potential security flaws during requirements definition, architecture, and high-level design [44] [9]. Due to a frequently evolving nature of defects, current-generation tools often fall short on being able to efficiently detect defects. Hence, it is important that detection methods continually evolve as well to keep up that pace [44]. Throughout literature, many studies have been conducted to explore, deduce, acknowledge, and establish ways to identify cybersecurity vulnerabilities in software, their causes, preventative measures, and effective methods of combating them. Some of the most common attempts to address security issues that are used by security experts and requirement engineers include approaches such as UMLSec, misuse cases, abuse cases, KAOS (Knowledge Acquisition in autOmated specification), SQUARE (security quality requirements engineering) and security patterns [9]. Apart from these traditional approaches, there are studies demonstrating techniques to utilize knowledge, experience, and expertise to capture and rectify security concerns during the early stages of software development. Through their study, Busby-Earle and France presented one such technique which captured, engaged, utilized, and refined knowledge of security experts and shared it amongst software engineering practitioners and requirements engineers [9]. This way authors helped assist them in their task of considering security concerns at the earliest stages of software development and then captured their expertise in the form of a type of dependency among requirements [9]. This technique, when combined with a less subjective approach, could reveal potential software vulnerabilities that can enhance those that require human expertise [9].

However, often addressing security vulnerabilities require specialized tools and techniques in addition to utilization of knowledge, experience, and expertise. Since, each software system or application is different from one another in terms of their domains and underlying architecture, each one of them require different approaches, practices, tools, and techniques for identifying and resolving security vulnerabilities during requirement elicitation, analysis, and documentation phases of requirements engineering. For instance, healthcare applications being a separate domain from web-services systems, each would require different techniques and approaches to incorporate security requirements during requirements engineering phase. For instance, Alzahrani et al. found Near Field Communication (NFC) technology and its security requirements to provide promising advantages for healthcare and medical service domain applications [1]. In their study, they used a multi-dimensional representation to propose a new classification of NFC system attacks in order to explore the possible threats to NFC systems in healthcare applications and the vulnerabilities and challenges it entails [1].

Their classification considered three main perspectives viz. mode of operation, programmability level and life cycle which were aimed to help developers identify the weakest protected point in their NFC system and use it as basis to study and evaluate the level of their NFC system security [1]. Gutiérrez et al. conducted a real web service-based case study and developed and presented an application of the process for Web Service Security (PWSSec) with the aim of eliciting and designing security in web service systems [19]. Yu et. al. also analyzed security vulnerability in web services software systems and attempted to map common attack patterns to security verification requirements [64].

3 Identifying Security Vulnerabilities in Design

Software vulnerabilities are usually not considered until very late in a software development life cycle which is why it is natural for them to live through the design phase unnoticed. By not discovering, identifying, assessing, and remediating vulnerabilities early in design activities, puts software applications into great risks. Design phase is an important phase in SDLC since it forms the basis for both software development and re-development of legacy systems. This is why design phase is the most susceptible to vulnerabilities as opposed to other phases of SDLC [50]. In his study on factors enabling security effectiveness, Savola mentioned that “software vulnerabilities originate from design-level flaws and implementation bugs, both manifested as exploitable faults, which, in turn, represent vulnerabilities” [53]. Hence, a good understanding of causes, measures and mitigation of vulnerabilities is crucial in the design phase. The causes of security vulnerabilities that are introduced in design stage can be categorized as – Cryptographic protocol design flaws, Relying on non-secure abstractions, Trading off security for convenience or functionality, No logging and, Design not capturing all risks [44]. Cryptographic protocol design flaws emerge as result of faults while designing cryptographic protocols. These protocols are especially hard to be correctly designed and takes several design iterations to get to a correct protocol [44]. Relying on non-secure abstractions forms the second category of causes of software vulnerabilities which constitute complexity-hiding abstractions more than tamper-proof abstractions that are offered by a programming language or by an operating system [44]. These abstractions lead to security breaches if they are implicitly or explicitly assumed to be tamper-proof [44]. Trading off security for convenience or functionality forms the third category of causes of software vulnerabilities which focuses on trade-offs between functionality or user-friendliness of the software i.e. trading-off between security and convenience [44]. While developing a software, its security aspects can be taken care of if equal amounts of attention is given to making things impossible while thinking about making things possible and convenient [44]. No logging forms the fourth category of causes of software vulnerabilities which emphasizes on missing efforts in detection and prevention of security breaches. It happens when designers spend more time thinking about preventive measures for security as opposed to spending time thinking about detection of security breaches. Creating and maintaining a good logging mechanism for following up on security incidents can help maintain a good balance between prevention and detection. [44]. Design does not capture all risks forms the last category of causes of software vulnerabilities which organizes all those cases where risks were identified early in the analysis phase but were never addressed in the design phase [44].

A successful identification of software vulnerabilities goes beyond specialized knowledge and expertise of software development, security, and business professionals. For years, researchers have constantly pondered upon, invented, discovered, and illustrated ways, approaches, tools and techniques towards thorough identification, mitigation, and prevention of software vulnerabilities from exploitation and attacks. To identify vulnerabilities in a system, Gegick and Williams matched a sequence of components in the design of a system that permitted the sequence of events in the attack pattern to occur [16] [17]. They called this methodology SAFE-T (Security Analysis for Existing

Threats) and proposed it for the purposes of starting security in the design phase while providing a taxonomy of attack patterns that describe security problems [16] [17]. Existence of a match helped them determine the existence of vulnerability in the application which is being analyzed. Authors concluded that conducting matching process in the design phases results in increased security awareness and encourages early beginning of risk management activities leveraging security team to determine ways to strengthen their application [16] [17]. Goseva-Popstojanova and Perhinschi recognized static analysis of source code to be a scalable method for discovering security vulnerabilities and software faults and focused their research work on empirical evaluation of the ability of tools that use static code analysis for detecting security vulnerabilities [45].

Yifan and Boonsiri also recognized the usefulness of pattern testing and proposed a method for detecting security vulnerabilities during design phase [62]. In their approach, they used model testing method to simulate attacks in UML simulation environment in accordance to the misuse patterns which helped uncover existing vulnerabilities that could expose a system to potential threats of certain types of attacks [62]. In addition to that, their method could also indicate which security patterns would be best suited to mitigate such risks. The benefits reaped from the contribution of model testing helped them analyze system security vulnerabilities during the design phase of software development [62]. Moreover, authors believed this method to be cost-effective and valuable since it had the potentials to improve the design resulting in lower costs as opposed to costs incurred during testing, debugging and maintenance phases of software development [62].

In an attempt to combat SQL injection vulnerabilities, Ciampa et. al. also utilized pattern matching approach and proposed a tool named V1p3R (“viper”) for penetration testing of web applications using standard SQL injections [11]. Their approach relied on an extensible knowledge base of heuristics which guided generation of SQL queries and was based on outputs produced by the pattern matching of error messages or valid outputs produced by the web application under test [11]. Other than pattern matching, researchers deduced other successful methods, approaches, algorithms, tools, and techniques to detect and resolve SQL injection vulnerabilities and attacks [34] [6] [51] [12] [65] [49] [4] [63] [30] [27] [29] [55]. The method proposed by Manmadhan and Manesh for preventing SQL injection attacks in JSP web applications involved checking of query semantics before executing [35]. Their detection approach was based on benign query generation from the final SQL query that was generated by the application and the inputs from the users and then comparing those semantics of safe query with that of the SQL query. Their main focus was on the stored procedure attacks wherein the query is formed within the database making it difficult for the query structure to be extracted before actual execution [35]. The SQL injection detection and prevention method proposed by Latha and Ramaraj involved manipulation of input attributes of the SQL query and measuring the distance of query strings [31]. Their method satisfied query analysis for both static and dynamic manipulation of user queries for several types of attacks including piggyback queries, tautologies, union queries, stored procedures, untyped parameters and inference alternate encodings [31].

4 Trust Modelling, Reputation, and Security

Security and Trust go hand in hand. A vulnerable software system which is more prone to exploitation by security attacks, is also likely more inclined towards loosing trust. Inclusion of security requirements is often ignored during the requirements elicitation and analysis phase of the SDLC and is then found to be obliviously falling under the non-functional requirements. After not being taken care of until very late in the development life cycle of a software application product, security vulnerabilities become hard to be identified and curbed. When security vulnerabilities, loopholes and workarounds are discovered by hackers and malicious attackers after the software product is delivered to the client, it lifts their trust off the software resulting in lesser usage and diminishing adaptability.

To successfully overcome security vulnerabilities and failures, it is important to gather, elicit and document security requirements in the earlier phases of software development, preferably, during the requirements engineering phase. Importance of security requirements and its engineering has been widely identified and accepted by researchers across literature. One of the earliest recognitions and breakthrough of security requirements was the development of SQUARE (Software Quality Requirements Engineering) methodology by Carnegie Mellon University's Software Engineering Institute's Networked Systems Survivability (NSS) Program [36]. The SQUARE process involved a team of requirements engineers and stakeholders of a project to interact and agree on technical definitions which would serve as a baseline for future communications [36]. This was followed by outlining of business and security goal. Then after, creation of artifacts took place which was necessary for a complete understanding of the relevant system [36]. After this, depending on several factors, including expertise of requirements engineering team, stakeholders involved, and the complexity and size of the project, a best method for eliciting initial security requirements from the concerned stakeholders was determined by the requirements engineering team. Upon the establishment of a method, the participants relied on the artifacts and the results of the risk assessment to elicit an initial set of security requirements [36]. The next two subsequent stages categorized and prioritized these requirements so that management and can use those towards making tradeoff decisions. As a conclusion, there was an inspection stage which ensured the accuracy and consistency of the generated security requirements [36]. To further understand the context and importance of security requirements, Haley et. al. presented a framework for the elicitation and analysis of these requirements. Their framework was based on construction of a context for the systems which represented security requirements as design constraints and developed satisfaction arguments for the security requirements which helped established the correctness of these requirements. Through their framework, they also explained the effects of security requirements on the functional requirements [21] [22]. Other approaches for eliciting and analyzing security requirements utilized usage of misuse cases [57] and security standards such as Common Criteria (ISO/IEC 15408) [37] [38].

Eliciting, analyzing, documenting, and incorporating these security requirements throughout the process of software development can help ward off security vulnerabilities and potential security attacks. Not only that, since security requirements ensure that security shall not be taken as a non-functional requirement or an afterthought, software development teams can use these to their advantage of staying abreast with latest security vulnerabilities and attack strategies. Integration of trust and security and benefits of trust modelling have been identified by researchers throughout literature [18]. Effective security design and implementation can be achieved by building security into every layer of a solution which can be accomplished by the usage of trust models [2]. Usage of acceptable trust models enable security architecture to provide a framework for delivering security mechanisms. "Trust modeling is the process performed by the security architect to define a complementary threat profile and trust model based on a use-case-driven data flow analysis" [2]. Information about the vulnerabilities, threats, and risk in an architecture of a particular information technology is integrated as a result of it [2]. With the help if trust modeling, specific mechanisms that are required to respond to a specific threat profile can be identified. "A threat profile is the set of threats and vulnerabilities identified through a use-case-driven data flow analysis that is particular to an organization. A threat profile identifies likely attackers and what they want" [2]. Depending upon the threshold for risks, trust relationships can be modeled using direct trust [2], transitive trust [2], assumptive trust [2] [20] and graphical tools [46]. Apart from studies on integration of trust and security and defining and incorporating security requirements into software development, concepts of zero trust cybersecurity models have also been proposed and explained [39] [48].

5 Security in Software Engineering and in Design Science

Analysis and Design phases of software development life cycle (SDLC) works in close conjunction with design science research methodologies and share the same goal of meeting technical requirements and business needs. It does so by creating artificial systems by building and evaluating artifacts [42].

Design science research process is iterative in nature which allows an artifact to emerge and opportunistically evolve as a solution or innovation. These artifacts include, and are not limited to, algorithms, methods [56], constructs, instantiations, models [42], human/computer interfaces, languages or notations, system designs, patterns, guidelines, requirements, and metrics, etc. [56]. Evaluation of these artifacts helps validate and provide attention towards usefulness, completeness, clarity, and rigor of an artifact [56]. Evolution of an artifact as a solution, so it can adapt and evolve through evaluations and implementations, can be achieved by its continuous evaluations against a set of rules and having it go through successive iterations [23]. This iterative nature of design science research also helps in capturing design issues, bugs, and defects during the early phases of analysis and design activities in turn reducing the overall project development timelines and repetitive product design efforts while improving overall product quality. It also ensures that all off the specified technical requirements have been met. Refining of artifacts in the design phase of SDLC helps designers and architects ensure the overall quality of to-be-developed software product in terms of its robustness, reliability, dependability, portability, correctness, availability, scalability, and integrity. Throughout literature, researchers have demonstrated explanatory and predictive knowledge of design science research to enhance construction of artifacts towards the betterment of software development [7] [24].

While design science research works with artifact iterations, software engineering employs design patterns for effective and efficient development of software applications. Design patterns offer reusable solutions to problems and issues arising in software design. They are like micro-architectures contributing to overall system architecture and serving as building blocks towards the construction of complex designs [15]. They capture intent behind a design to preserve information and identify roles of classes and instances, distribution of responsibilities and collaborations [15]. They help designers explore design alternatives, communicate and document them by providing them with a common vocabulary [15]. Since design patterns name and define abstractions above classes and instances, it helps reduce system complexity [15]. Design patterns help with refactoring or reorganization of class hierarchies and their usage early in the lifecycle can deter refactoring at the later stages of design [15]. This can also help with taking care of missing requirements, significantly reducing costs associated with rework activities and occurrences of design defects and lowering total expected program costs in turn improving the quality of software product [59].

The qualities and working of design patterns are similar in nature with design science research methodologies. Design science research advocates iteration of artifacts to improve design process, similar to structuring, reorganization, and refactoring abilities of design patterns. This iteration process helps identify design issues and defects early in the design process. Likewise, usage of different design metrics can further measure and evaluate the quality software designs contributing towards reduced number of defects during software development [47].

Parallels between design science and software engineering in terms of security can be drawn over the concept of security patterns. Since, there is a lack of mechanical aids to detect design errors, assessing security at design level is difficult [54]. Security patterns offer solutions for these security problems. They utilize the structure provided by patterns to provide guidelines for secure system design and evaluation by joining the extensive knowledge accumulated about security [14]. These patterns not only provide value to new system designs but are also useful in evaluating existing systems. Moreover, they also help compare security standards and verify whether the product complies with some standard [14]. Security patterns stop or mitigate attacks by describing mechanisms that fall into any of the security countermeasures categories such as Identification and Authentication, Logging, Access Control and Authorization, Cryptography and Intrusion Detection [14].

Throughout literature, researchers have studied security patterns, enhanced them, demonstrated their conformance to security standards and utilized them in establishing security requirements for the software to-be-developed. Konrad et. al investigated and presented a tailored template for security patterns to address difficulties that are integral during the development of security-critical systems [28]. Their template included identification of security-specific consequences of applying a pattern, formal specification of security-oriented constraints, and an explicit identification of security-specific development principles to be addressed during the application of a given pattern [28]. They also made use of UML notations for representing structural and behavioral information which can further maximize comprehensibility of their template [28]. These patterns can help developers that are new to addressing security issues during development with gaining insights into the modeling and analysis of security concerns starting from the requirements engineering phase [28]. Another approach for the selection of security patterns for meeting security requirements was presented by Weiss and Mouratidis [61]. Their approach helped enable an in-depth understanding of the trade-offs that the patterns involve and the implications of the pattern to several security concerns [61].

6 Conclusion

Software engineering principles and alternative fields of design science methodology, trust, and reputation, being independent domains in their own, still share a common relationship with the underlying principles of cybersecurity. Security vulnerabilities, their identification, categorization, detection, prevention, mitigation and remediation ties in with strategies, approaches, tools and techniques that are offered by trust modelling and design science research methodology. Importance of understanding and combating security vulnerabilities by inclusion of security requirements at requirements elicitation and analysis and design phases has been studied in this paper. A thorough illustration of identification, categorization of security issues along with approaches to remedy them was described. Lastly, connection between software engineering and design science were explored.

References

[1] Ali Alzahrani, Abdullah Alqhtani, Haytham Elmiligi, Fayez Gebali, and Mohamed S. Yasein. NFC security analysis and vulnerabilities in healthcare applications. *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 302-305, Victoria, BC, Canada, August 27-29, 2013.

[2] Donna Andert, Robin Wakefield and Joel Weise. *Trust modeling for security architecture development*. Sun MicroSystems BluePrints, 2002.

[3] J. Antunes, N. Neves, M. Correia, P. Verissimo and R. Neves. Vulnerability Discovery with Attack Injection. *IEEE Transactions on Software Engineering*, 36(3):357-370, 2010.

[4] Jalal O. Atoum and Amer J. Qaralleh. A Hybrid Technique for SQL Injection Attacks Detection and Prevention. *International Journal of Database Management Systems*, 6(1):21-28, 2014.

[5] Steven Baker. Fuzzing: A Solution Chosen by the FDA to Investigate Detection of Software Vulnerabilities. *Biomedical Instrumentation & Technology*, 48(1):42-47, 2014.

[6] Yogesh Bansal and Jin H. Park. Multi-hashing for Protecting Web Applications from SQL Injection Attacks. *Int Journal of Computer and Communication Engineering*, 4(3):187-195, 2015.

[7] Roman Beck and Sven Weber. Enhancing IT Artifact Construction with Explanatory and Predictive Knowledge in Design Science Research. *Journal of Information Technology Case and Application Research*, 15(1):4-18, 2013.

[8] Sofia Bekrar, Chaouki Bekrar, Roland Groz, and Laurent Mounier. Finding software vulnerabilities by smart fuzzing. *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 427-430, Berlin, Germany, March 21-25, 2011.

[9] Curtis C. Busby-Earle, and Robert B. France. Analysing Requirements to Detect Latent Security Vulnerabilities, San Francisco, California, USA, June 30 – July 2, 2013.

[10] Steve Christey and Robert A. Martin. Vulnerability type distributions in CVE, 2007.

[11] Angelo Ciampa, Corrado A. Visaggio and Massimiliano Di Penta. A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, pages 43-49, Cape Town, South Africa, May 02, 2010.

[12] Debasish Das, Utpal Sharma and D. K. Bhattacharyya. Rule based Detection of SQL Injection Attack. *International Journal of Computer Applications*, 43(19):15-24, 2012.

[13] Yuri Demchenko, Leon Gommans, Cees de Laat and Bas Oudenaarde. Web services and grid security vulnerabilities and threats analysis and model. *Procs of the 6th IEEE/ACM international workshop on grid computing*, pages 262-267, Washington, DC, USA, November 13-14, 2005.

[14] Eduardo B. Fernandez, Hironori Washizaki, Nobukazu Yoshioka, Atsuto Kubo and Yoshiaki Fukazawa. Classifying security patterns. *Asia-Pacific Web Conference*, pages 342-347, Shenyang, China, April 26-28, 2008.

[15] Erich Gamma, Richard Helm, Ralph Jonhson and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. *European Conference on Object-Oriented Programming*, pages 406-431, Kaiserslautern, Germany, July 26-30, 1993.

[16] Michael Gegick and Laurie Williams. Matching attack patterns to security vulnerabilities in software-intensive system designs. *ACM SIGSOFT Software Engineering Notes*, 30(4):1-7, 2005.

[17] Michael Gegick and Laurie Williams. On the design of more secure software-intensive systems by use of attack patterns. *Information and Software Technology*, 49(4):381-397, 2007.

[18] Paolo Giorgini, Fabio Massacci, John Mylopoulos and Nicola Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *International Journal of Information Security*, 5(4):257-274, 2006.

[19] Carlos Gutiérrez, David Rosado and Eduardo Fernández-Medina. The practical application of a process for eliciting and designing security in web service systems. *Information and Software Technology*, 51(12):1712-1738, 2009.

[20] Charles B. Haley, Robin Laney, Jonathan D. Moffett and Bashar Nuseibeh. Using trust assumptions with security requirements. *Requirements Engineering*, 11(2):138-151, 2005.

[21] Charles B. Haley, Jonathan D. Moffett, Robin Laney and Bashar Nuseibeh. A framework for security requirements engineering. *Proceedings of the 2006 international workshop on Software engineering for secure systems*, pages 35-42, Shanghai, China, May 20-21, 2006.

[22] Charles B. Haley, Jonathan D. Moffett, Robin Laney and Bashar Nuseibeh. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*, 34(1):133-153, 2008.

[23] Marlien Herselman and Adele Botha. Evaluating an artifact in design science research. *Proceedings of the 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists*, pages 1-10, Stellenbosch, South Africa, September, 28-30, 2015.

[24] Alan R. Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):87-92, 2007.

[25] Jungho Kang and Jong H. Park. A secure-coding and vulnerability check system based on smart-fuzzing and exploit. *Neurocomputing*, 256:23-34, 2017.

[26] J. Kim, Y. K. Malaiya and I. Ray. Vulnerability discovery in multi-version software systems. *10th IEEE High Assurance Systems Engineering Symposium, HASE'07*, pages 141-148, Plano, Texas, USA, Novermber 14-16, 2007.

[27] Abhay K. Kolhe and Pratik Adhikari. Injection, Detection, Prevention of SQL Injection Attacks. *International Journal of Computer Applications*, 87(7):40-43, 2014.

[28] Sascha Konrad, Betty H. C. Cheng, Laura A. Campbell and Ronald Wassermann. Using security patterns to model and analyze security requirements. *2nd International Workshop on Requirements Engineering for High Assurance Systems (RHAS '03)*, pages 13-22, Monterey Bay, California, USA, 2003.

[29] Niraj Kulkarni, D. R. Anekar, Mayur Ghadge and Rohit Garde. Multi-Agent System for Detecting and Blocking SQL Injection. *Int Journal of Computer Applications*, 64(15):42-45, 2013.

[30] Z. Lashkaripour and A. G. Bafghi. A Simple and Fast Technique for Detection and Prevention of SQL Injection Attacks (SQLIAs). *International Journal of Security and Its Applications*, 7(5):53-66, 2013.

[31] R. Latha and E. Ramaraj. SQL Injection Detection Based On Replacing The SQL Query Parameter. *International Journal Of Engineering And Computer Science*, 2015.

[32] Tong Li, Xuan Huang and Huang Rui. Research on Software Security Vulnerability Discovery Based on Fuzzing. *Applied Mechanics and Materials*, 635-637: 1609-1613, 2014.

[33] Hongzhe Li, Hyuckmin Kwon, Jonghoon Kwon and Heejo Lee. CLORIFI: software vulnerability discovery using code clone verification. *Concurrency and Computation: Practice and Experience*, 28(6):1900-1917, 2015.

[34] Ouarda Lounis, Salah E. B. Guermeche, Lalia Saoudi and Salah E. Benaicha. A new algorithm for detecting SQL injection attack in Web application. *International Journal of Advanced Computer Science and Applications*, 4(3):43-51, 2014.

[35] Sruthy Manmadhan and Manesh T. A Method of Detecting Sql Injection Attack to Secure Web Applications. *International Journal of Distributed and Parallel systems*, 3(6):1-8, 2012.

[36] Nancy R. Mead, Eric D. Hough and Theodore R. Stehney II. Software Quality Requirements Engineering (SQUARE) methodology, *SESS '05 Proceedings of the 2005 workshop on Software engineering for secure systems—building trustworthy applications*, pages 1-7, St. Louis, Missouri, USA, May 15-16, 2005.

[37] Daniel Mellado, Eduardo Fernández-Medina and Mario Piattini. A common criteria based security requirements engineering process for the development of secure information systems. *Computer Standards & Interfaces*, 29(2):244-253, 2007.

[38] Daniel Mellado, Carlos Blanco, Luis E. Sánchez and Eduardo Fernández-Medina. A systematic review of security requirements engineering. *Computer Standards & Interfaces*, 32(4):153-165, 2010.

[39] Robert J. Michalsky. Zero Trust Model Cyber Security Explained. http://www.njvc.com/blog/cyber-security/zero-trust-cyber-security-model-understanding, Aug. 2015.

[40] Maryam Mouzarani, Babak Sadeghiyan and Mohammad Zolfaghari. A smart fuzzing method for detecting heap-based vulnerabilities in executable codes. *Security and Communication Networks*, 9(18):5098-5115, 2016.

[41] Ian Muscat. Web vulnerabilities: identifying patterns and remedies. *Network Security*, 2016(2):5-10, 2016.

[42] Lukasz Ostrowski, Markus Helfert and Nelson Gama. Ontology engineering step in design science research methodology: a technique to gather and reuse knowledge. *Behaviour & Information Technology*, 33(5):443-451, 2014.

[43] J. Pescatore. Taxonomy of software vulnerabilities. *The Gartner Group Report*, 2003.

[44] Frank Piessens. A taxonomy of causes of software vulnerabilities in internet software. *Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering*, pages 47-52, Annapolis, Maryland, USA, November 12-15, 2002.

[45] Katerina Goseva-Popstojanova and Andrei Perhinschi. On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*, 68: 18-33, 2015.

[46] Steve Purser. A Simple Graphical Tool For Modelling Trust. *Computers & Security*, 20(6):479-484, 2001.

[47] M. R. J. Qureshi and Waseem A. Qureshi. Evaluation of the Design Metric to Reduce the Number of Defects in Software Development. *International Journal of Information Technology and Computer Science*, 4(4):9-17, 2012.

[48] Aishwarya Rane. Zero Trust Model for Optimum Cyber Security. https://simplesnippets.tech/zero-trust-model-for-cyber-security/, June 2018. Available: https://simplesnippets.tech/zero-trust-model-for-cyber-security/amp/. [Accessed: 28- Jul- 2018].

[49] Romil Rawat and Shailendra Shrivastav. SQL injection attack Detection using SVM. *International Journal of Computer Applications*, 42(13):1-4, 2012.

[50] S. Rehman and K. Mustafa. Research on software design level security vulnerabilities. *ACM SIGSOFT Software Engineering Notes*, 34(6):1-5, 2009.

[51] Sangita Roy, Avinash K. Singh and Ashok S. Sairam. Detecting and Defeating SQL Injection Attacks. *International Journal of Information and Electronics Engineering*, 1(1):38-46, 2011.

[52] P. Salini and S. Kanmani. Survey and analysis on Security Requirements Engineering. *Computers & Electrical Engineering*, 38(6):1785-1797, 2012.

[53] Reijo M. Savola. Towards Measurement of Security Effectiveness Enabling Factors in Software Intensive Systems. Lecture Notes on Software Engineering, 2(1):104-109, 2014.

[54] Markus Schumacher. Security Patterns and Security Standards. European Conference on Pattern Languages of Programs, 289-300, 2002.

[55] Lwin K. Shar and Hee B. K. Tan. Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Information and Software Technology*, 55(10):1767-1780, 2013.

[56] Nur S. A. Shukor, Azizah A. Rahman, Noorminshah A. Iahad. Summative Evaluation for Design Science Artifact using Structured Walkthrough. *2017 International Conference on Research and Innovation in Information Systems*, pages 1-6, Langkawi, Malaysia, July, 16-17, 2017.

[57] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34-44, 2004.

[58] B.R. Singh. Vulnerability discovery with attack injection software vulnerability discovery. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(9):451-456, 2013.

[59] James J. Y. Tan, Kevin N. Otto and Kristin L. Wood. Relative impact of early versus late design decisions in systems development. *Design Science*, 3(12), 2017.

[60] John Viega, Tom Mutdosch, Gary McGraw and Edward W. Felten. Statically Scanning Java Code: Finding Security Vulnerabilities. *IEEE Software*, 17(5):68-74, 2000.

[61] Michael Weiss and Haralambos Mouratidis. Selecting Security Patterns that Fulfill Security Requirements. *16th IEEE International Requirements Engineering Conference*, pages 169-172, Catalunya, Spain, September, 8-12, 2008.

[62] Yuan Yifan and Somjai Boonsiri. Analysis of Security Vulnerabilities Using Misuse Pattern Testing Approach. *Journal of Software*, 10(5):650-658, 2015.

[63] Gülsüm Yiğit and Merve Arnavutoğlu. SQL Injection Attacks Detection & Prevention Techniques. *International Journal of Computer Theory and Engineering*, 9(5):351-356, 2017.

[64] Weider D. Yu, Dhanya Aravind and Passarawarin Supthaweesuk. Software vulnerability analysis for web services software systems. *11th IEEE Symposium on Computers and Communications*, pages 740-748, Cagliari, Italy, June, 26-29, 2006.

[65] Zongzhi Zhang, Qiaoyan Wen and Zhao Zhang. An Improved Approach for SQL Injection Vulnerabilities Detection. *Applied Mechanics and Materials*, 263-266:3017-3020, 2012.