OPEN ACCESS Freely available online

Information Technology & Software Engineering

Review Article

A Review of Past Software Strategy and Secure Software Engineering for Modern Cybernetics

Faisal Nabi*

Department of Information and Cybersecurity, University of Southern Queensland, Toowoomba, Australia

ABSTRACT

Secure software engineering is always a subject of debate among researchers and scientists. Some of have experience with solutions through life cycles some of the findings originated from attacks by proposing software engineering-based taxonomy. This paper considers the past and present challenges of secure software engineering. We will review all aspects of software engineering and then evaluate the right path from the present to the future of secure software engineering. Highlight the origin of flaws, errors and faults in software and how can we capture it in the initial stages of lifecycle of software development.

Keywords: Software; Security; Flaw in design; Privacy; Secure software; SDLC security

INTRODUCTION

Take into account the situation of a manager or technical lead who is in charge of creating a new "connected" software product, such as a server application for the Web.

This linked device will leverage Internet infrastructure and/or technology to fulfil one or more of its requirements. The topic of security comes up early in the development cycle, perhaps during the specification of requirements. The management has been primarily focused on functional requirements and features, while security has received minimal attention. This fictitious situation raises a number of pertinent issues that have direct implications for Secure Software Engineering (SSE):

- Are we really going to be attacked? Could an attack on the software actually happen?
- Aren't network administrators responsible for security? How do the vulnerabilities in the software we're creating get opened?
- Existing solutions that we could enhance to offer the required defences?
- Should we be alarmed? What impact would a security breach have on our company, our products, and/or our clients?
- What security risks should we try to reduce as a development team? What steps should we take to create secure software?
- How can we tell if we're successfully enhancing the security of our software?

These six questions are due to insecure process of software coding, design and reading requirement specification as compare to function presented to customers and solution finders. The questions are based on actual interactions the author has had with managers and other developers; the situation is made up. Furthermore, works like confirm the validity of the aforementioned queries and their industry wide generalizations. A software engineering manager or lead developer should be better prepared to respond to the questions above after reading this article. This paper's main contribution is to compile and analyze SSE research, making it more publicly available to an audience of practitioners, which is long needed (given the current state of affairs). A fundamental question is, “Will we really be attacked? Or, could the software really be at risk for attack?” First, this question should not be dismissed lightly, since at least theoretically any money, time, or other resources devoted to security will be completely wasted if you are never attacked. Second, this question is not trivial; it’s difficult to quantitatively determine attack “attractiveness.” Intuitively, factors such as connectivity, popularity, attack difficulty, and market penetration play a role in estimating how attractive your company or product is to would-be attackers, But it's impossible to directly link these elements to the likelihood of an assault (*i.e.*, used in risk analysis) [1-2] That being said, the assault probability is probably not 0.

Correspondence to: Faisal Nabi, Department of Information and Cybersecurity, University of Southern Queensland, Toowoomba, Australia; E-mail: faisal.nabi@yahoo.com

Received: 06-Dec-2022, Manuscript No. JITSE-22-20647; **Editor assigned:** 08-Dec-2022, PreQC No. JITSE-22-20647 (PQ); **Reviewed:** 22-Dec-2022, QC No. JITSE-22-20647; **Revised:** 12-May-2023, Manuscript No. JITSE-22-20647 (R); **Published:** 19-May-2023, DOI: 10.35248/2165-7866.23.13.334

Citation: Nabi F (2023) A Review of Past Software Strategy and Secure Software Engineering for Modern Cybernetics. J Inform Tech Softw Eng. 13:334.

Currently, there is a feature centered development culture that frequently pays little attention to security concerns, at least in the commercial sector [3].

To see that security has once again been neglected during the software development life cycle, one simply needs to glance at the never-ending stream of news headlines highlighting the most recent exploits (SDLC) The problem isn't that businesses don't care about security; rather, it's that managers and engineers don't know how software development activities relate to end-product security [4,5].

For instance, the US-CERT vulnerability bulletins and NIST national vulnerability database both strongly suggest that the right question is not instead of asking "Will I be attacked?" ask "How will I be attacked?"

If we've learned anything from the past, it should be that the software we create will be utilized in hostile contexts, frequently out of our control, where it will be stressed to the breaking point and employed in unexpected ways to achieve nefarious goals. This has never been more accurate than it is right now, and things aren't going any better, especially for connected software.

Although firewall technology, operating systems, and secure/encrypted connections have all seen significant advancements [6].

The software itself has been the primary cause of numerous security flaws. Experts expect that attackers will increasingly target the application layer directly due to advancements achieved in other areas such as operating systems, firewalls, and secure communications as well as the fact that attackers will take the easiest route [7].

Connectivity and technological convergence in embedded devices are other factors driving the rising interest in SSE.

Especially wireless phones. Numerous desktop programmes, including web browsers, email, media players, and instant messaging services, are integrating with mobile platforms. Numerous of these programmes have ongoing, widely known security problems. The prevalence of wireless phone users and the convergence of venerable desktop functions afflicted by security need the specification, design, and implementation of these features. More thoroughly tested to prevent the widespread security vulnerabilities that afflict desktop systems at the moment.

Software security is always a burning issue and need such a consideration to protect of its early stage of requirement engineering to process of SDLC level and need to reconsider SDLC and need to point out stages of security and insecurity. Although the material and concepts presented are generally applicable, this article is primarily aimed for software project managers and developers in the commercial sector. To be more precise, the consumer electronics and consumer software industries were taken into consideration when writing this essay. This paper's main contribution is to compile and analyse SSE research, making it more publicly available to an audience of practitioners, which is long needed (given the current state of affairs).

What SSE is and is not is defined in this section

Simply put, developing secure software does not always include developing security software. SSE aims to use procedures, principles, and techniques to create software that is devoid of vulnerabilities and always maintains a secure state under attack but still manages to serve authorized users. As was already said, SSE does not always involve designing security software. Although SSE techniques can be used to create security software, they can also constructing any kind of programme. Additionally, as will be covered momentarily, it is becoming more crucial to consider using SSE approaches with application software.

Software faults always cause of cybersecurity, this is another reason of software security this needs to revive. It is important to review cybersecurity issues, attacks and fault hoop wholes. We provide first technique and definition and then covers the single software error, fault flaw to failure of system based software applications.

The goal of cyber-security is to protect networks, computers, programmes, and data from unauthorized access and attacks. This section introduces cyber-attacks and provides terminology for them. Describing cyber-threats and cyber-security systems. The second section proposes a classification of cyber-security mechanisms that will be useful in assessing their impact on privacy.

Cyber-attack classification

The goal of the attack is a first dimension for classifying an attack. This is frequently related to how an adversary monetizes the attack (for example, by stealing information and selling it to advertisers or other third parties) criminals. Overall, the attack objectives are classified as one of the following:

- Stealing information, such as device data, media files, and user credentials; this action is typically carried out by spyware malware
- Tracking user information, , monitoring users' sensitive data (e.g., locations, activities, or health-related data); this action is typically carried out by mobile malware
- Gaining control of a system, as trojans, botnets, and rootkits do.

LITERATURE REVIEW

Structure of paper and objective

Following is an outline for this paper. Following a detailed definition of vulnerabilities, a description of their causes and how they relate to threats, and a discussion of ongoing classification efforts, a high-level overview of the SSE field and its associated issues is provided. Threat and vulnerability relationships can be utilized to specialize a conventional operational risk assessment calculation for software, as shown by the discussion of threats and vulnerabilities into the issue of risk analysis for computer and network security in general.

Research background

Many software engineers have a propensity to see computer and network security as an operational concern that the IT

department or network administrators manage (even those with years of experience) [8]. In the IT industry, by setting up firewalls, updating virus definitions, and using the most recent patches, security assurance can be attained. Keep in mind that firewalls and Patches, according to Hoglund and McGraw, are only "band-aid" fixes for software flaws. The software development context is the ideal framework for addressing the primary cause of the majority of computer security failures.

A nearly automatic response when security is brought up in the context of development is to consider specific security features like cryptography, authentication, and copy protection and how the development team may incorporate or add software components that offer these security capabilities. Because many of these components:

- Frequently employ advanced and tested encryption.
- Frequently already exist, there is a propensity to use add-on security components.

Given by the underlying Operating System (OS) platform, packed, or packaged. Simply put, it makes financial sense to reuse such components in terms of cost and security assurance.

The development team must understand that these supplemental elements are not a security panacea, though. Simply "bolting-on" security software will not provide security assurance. The specifications, design, code, and chosen security may be affected by implementation language in general.

Multiple software categories collaborate in a complete system, such as a web server, and are packaged in several logical components. And frequently arranged in layers (e.g., OS/Kernel/Driver, network, and application). An opportunity for an attacker arises from a vulnerability in any component at any layer. Any layer of the system is vulnerable to attack, and frequently the weakest layer will be the target. In fact, since much research has led to security improvements in the operating system, networking, and cryptography layers, attackers are increasingly targeting the application layer. Secure software engineering is concerned with engineering software (all types) such that the end product provides some level of security assurance. SSE is predicated on fact that attackers frequently exploit vulnerabilities originating within the Software Development Life Cycle (SDLC); during requirements, design, implementation, or are missed during verification [9]. Section 3 more deeply explores and defines the term vulnerability and associated high-level concepts.

The safe sector of software engineering is currently conducting research on security based on:

Current studies in the secure branch of software engineering concentrate on security focusing on:

- Developer training,
- Processes used in software development,
- Best practices,
- Cases of requirements and abuse,
- The classification and listing of threats,
- Design and architecture,
- Model based testing.

According to scholarly research that has already been done and that is still needed, it appears that many of our upcoming technical issues will be related to requirements, design, and measurements. Issues facing the commercial sector involve effectively applying academic research achievements (particularly those related to decreasing implementation level faults) in real world contexts.

According to scholarly research that has already been done and that is still needed, it appears that many of our upcoming technical issues will be related to requirements, design, and measurements. The business sector faces difficulties in effectively using completed academic research (mainly focused on eliminating implementation level faults) in practical ways.

The gap between academic advancements and the state of industry practice in all fields, and implementation in particular, is still very vast.

About 50% of all exploited vulnerabilities are implementation level flaws; with the remaining 50 percent resulting from needs and design flaws. Even if the implementation tools were more commonly used, only a small proportion of developers (in comparison to the majority) would be skilled in their use [10].

Vulnerability indifferent cause and shapes

Vulnerabilities result from defects. The defects may be easily identifiable, code level bugs resulting from implementation, or may result from more deeply seated issues/oversights (*i.e.*, flaws) in the design or requirements. The following list enumerates various phases of the SDLC and briefly highlights the ways in which vulnerabilities manifest:

Due to insufficient or frequently absent security centric requirements throughout the requirements definition phase. Requirements might not take malevolent and unanticipated uses into account, security requirements are frequently expressed as non-functional requirements, frequently mentioning a specific technology to utilize rather than identifying dangers and describing the threat environment. Unintentional/malicious applications of the software may not be addressed in subsequent stages of the software development life cycle, which is a coincidence.

Error handling, policy enforcement, and other security-related cross-cutting concerns were not sufficiently taken into account during the design phase, and as a result.

Component composition it's also possible that security wasn't even considered at all when designing something.

The implementation phase, which is caused by the use of non-type safe languages, unsecured Application Programming Interfaces (APIs), poor use of secure APIs, seeding by malicious developers, and a lack of secure coding practices, or simply insufficient developer education with regard to code-level flaws that attackers typically exploit.

Failure to detect vulnerabilities established in the requirements, design, and implementation phases during the verification/testing phase. Testing staff might lack the architectural expertise and/or malevolent abilities required to identify security flaws.

Even security testing technologies are now restricted to black-box assaults and may not work effectively with the system being tested.

The maintenance phase, which is brought on by erroneous deployment configurations or flaws that updates and bug patches cause. Since patches are often generated in maintenance, vulnerabilities with origins there have a lot in common with those with origins in earlier periods. Vulnerabilities might arise in many of the same places as they did in earlier SDLC phases because the patch is a software product.

Vulnerabilities can infiltrate the software in a variety of ways and at different stages. In general, the sooner in the SDLC the vulnerability is discovered.

The more deeply introduced (or entrenched in), the more challenging (and expensive) it would be to fix the associated flaw. For instance, it might be relatively simple to fix a one-line code problem, yet it would be nearly impossible to "apply" a "patch" in the classic sense to a flaw that was deeply ingrained in the design. In this situation, a total redesign might be necessary to fix the design problem.

Threats are relative to their target vulnerabilities, as the germ metaphor shows. Threats are a warning that an attacker might try to take advantage of a target's vulnerability. Figures 1 and 2 depicts a threat t, that aims at vulnerability, V that can be exploited.

Because exploited vulnerabilities are frequently localized around input/output interfaces and the software/system boundary (e.g., files, communication channels, and system resources like memory and CPU; it is important to note that the illustration places V close to the boundary.

DISCUSSION

Threats to software are only important to take into account when there are enabling vulnerabilities (or are likely to be present) in the intended use. Threats' potential for harm is correlated with the effects of exploited vulnerabilities. Threats that target the associated vulnerabilities cease to be a worry, and the programme is no longer at risk as a result of those specific threats. Figure 3 displays this relation. Remember that the vulnerability could be a typical code-level problem (like a buffer overflow) or a latent design flaw (perhaps an oversight spread from requirements) (Figures 1-3).

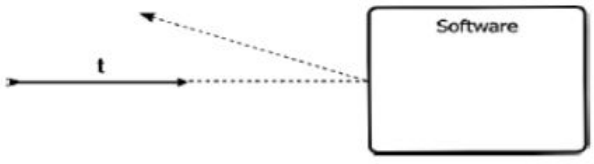

Figure 1: Threat target T vulnerability V.

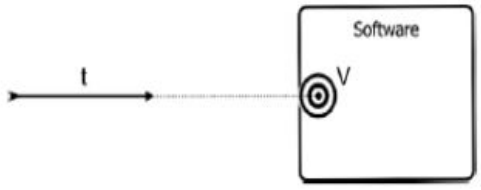

Figure 2: Software immune to threat t.

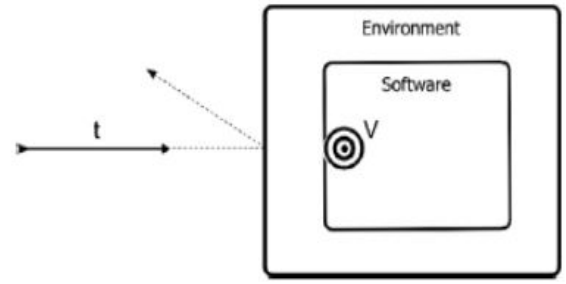

Figure 3: Environment shields vulnerability V from threat t.

This is the research work previously done to explain the vulnerability and its birth related work however this is incomplete work to exemplify the birth of vulnerability in software and design of software base applications.

The modern work of R and D on the software birth life cycle research is no for more advance.

Modern evidence of software vulnerability birth cycle

Current research is exemplified that design and flaws are logical issues of software function and requirements from beginning of its process life cycle. Therefore, need of comprehensive life cycle is desired from birth to final phase of product for developers and practitioners. There is need of a taxonomy based life cycle.

As stated in the introduction, our primary goal is to build a software taxonomy of logical weaknesses in distributed multiple-tier information systems' application layer. Various papers and texts have established a number of approaches to evaluate the security of technical infrastructure for information and communication, which serve as a springboard for software application.

In the component based software applications and systems, we propose the SVAM for the main computer protection attributes 'Five Columns,' as mentioned, showing the life cycle of the vulnerability and classifying the key point where the vulnerability covers two or more delicate vulnerability classes, such as 'Technical and Logical,' as defined in Figure 4 (Table 1).

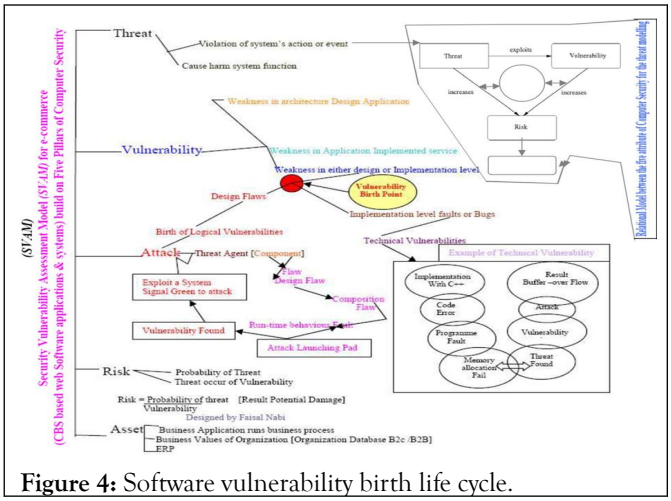

Figure 4: Software vulnerability birth life cycle.

Table 1: Attack pattern properties.

Pattern name and classification	A unique, descriptive identifier for the pattern
Attack prerequisites	What conditions must exist or what functionality and what characteristics must the target software has, or what behavior must it exhibit, for this attack to succeed?
Description	A summary of the assault including the course of action
Related vulnerabilities or weakness	What specific vulnerabilities or weakness.
Method of attack	Which sort of attack vector utilized (e.g. malicious data entry, maliciously crafted file, protocol corruption)?

- Coding faults are composed of faults in the software development process that are introduced during software development. These faults are the cause of errors in programming logic and missing or incorrect requirements.
- Operational faults operational faults are called incorrect software deployment. In most situations, failures can be categorized as operational faults according to Aslam, 1995.
- Environment faults occur when a programmer does not completely understand the limitations of the usable right modules or the interactions between them according to Krsul, 1998.
- The source of this taxonomy is based on Faisal Nabi 2021 research work, the given below is an overview of vulnerability and its causes (Figure 5).

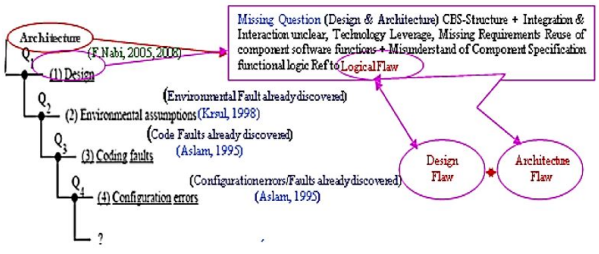

Figure 5: Taxonomy of software vulnerabilities causes 2021 Faisal Nabi, et al.

This section discussed the role of threats and vulnerabilities in risk assessment. As previously stated, risk assessment is not trivial, despite its conceptual simplicity. In practice due to the difficulty of quantifying traditionally qualitative and intangible factors (e.g., target 19 secure software engineering attractiveness and reputation loss). The definition and discussion of threats, attacks, and vulnerabilities, as well as the treatment of risk assessment as explained in Figure 4 and Figure 5 to completely explain the software defects and its birth lifecycle in the light of modern research.

A comprehensive vulnerability taxonomy is detailed here for researcher and practitioners (Figure 6).

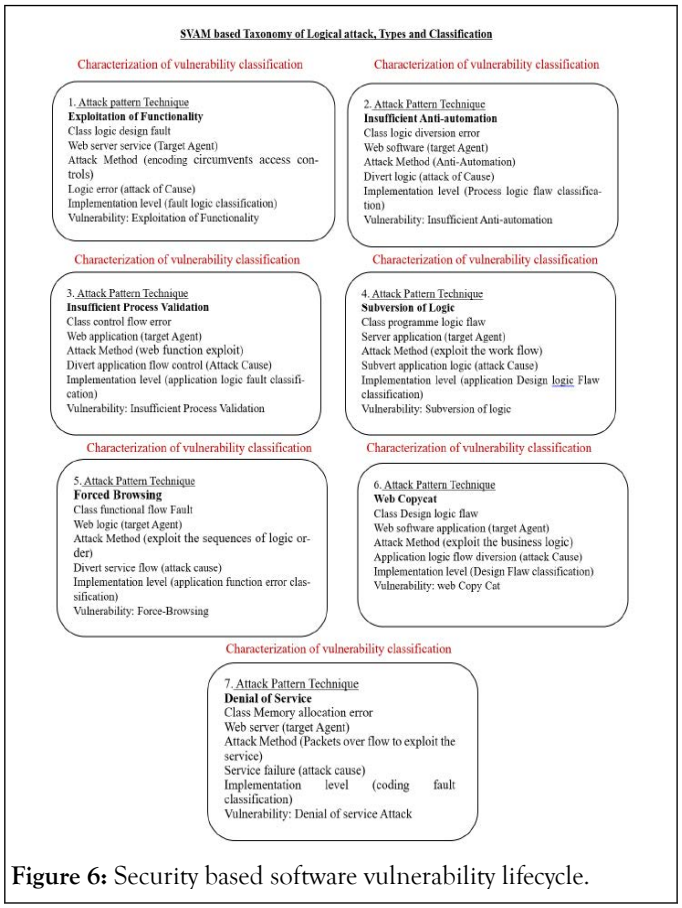

Figure 6: Security based software vulnerability lifecycle.

This taxonomy explains the details of software based causes and defects that may violates the boundary condition of a software and it application specially component based software applications.

In light of our findings, we propose a classification and characterization of the two categories of vulnerability problems/ issues mentioned above (technical *vs.* logical vulnerabilities). These are classified as previously stated in the classification of each weakness based on its attack process (attack pattern technique). As a result of retaining the classification of two distinct vulnerability types, we created a classification tree that includes all sub-class attacks within each vulnerability class. A new taxonomy with detailed classification is shown here, distinguished by its distinct signature in the application layer.

Analysis of problem based discussion

Prioritizing software entails calculating a variant of the "value of protection" formula. The formula for "value of protection" represents a traditional framework for assessing computer/network security risks, classified as a "financial loss methodology. The original formula is as follows:

$V_oP=R-M$, or

$V_oP=(A_p \times L)-M$

Where;

V_oP, is the value of protection (aka priority),

R, is the risk, calculated as (A_p x L),

A_p, is the probability of a successful attack,

L, is the loss resulting from a successful attack, and

M, is the cost of the mitigation countermeasure.

Applying the relationship among attack, threat, and vulnerability (discussed in the equation can be better adapted for software security risk analysis. Realizing that A_p can be rephrased as the probability that an attacker will successfully exploit vulnerability V to realize a given threat, and that threats target vulnerabilities, the formula has been modified to approximate threat probability.

CONCLUSION

This paper provided a high-level overview of SSE in an attempt to answer practical questions about the field. Keeping vulnerabilities from being introduc

The challenge SSE rises to meet is to patch vulnerabilities prior to release rather than after the fact. Outside of the context of formal methods and automated theorem provers (which typically necessitate significant investment and expertise), proving the security of a typical software product necessitates testing for negative consequences (e.g., there are no vulnerabilities). Because testing for a negative can entail indefinite testing time, it is critical to incorporate security from the start and make cost-effective decisions about when to stop testing.

A key overarching practical consideration is how to approach vulnerability prevention in a cost effective manner. This is not a novel thought.

SSE is addressed by risk analysis in general, which is based on the fact that some level of planning and up-front expenditure is required to obtain security assurance. In general, risk analysis must balance the upfront costs of vulnerability protection against the likelihood that attackers will target those vulnerabilities. The more precise the risk analysis, the better the contingent business decisions will be.

This paper discussed risk analysis aspects from previously studied fields, namely software engineering and computer/ network security, and provided an example of how the concepts translate to software.

REFERENCES

1. Warwick K. The promise and threat of modern cybernetics. South Med J. 2007;100(1):112-125.
2. Glanville R. Cybernetics: Thinking through the technology. In Trad Sys Theory. 2013;57-89.
3. Wang Y, Kinsner W, Zhang D. Contemporary cybernetics and its facets of cognitive informatics and computational intelligence. IEEE Trans Syst Man Cybern B Cybern. 2009;39(4):823-833.
4. Yang H, Chen F, Aliyu S. Modern software cybernetics: New trends. J Syst Softw. 2017;124:169-186.
5. Levit GS, Hossfeld U, Olsson L. From the "modern synthesis" to cybernetics: Ivan Ivanovich Schmalhausen (1884-1963) and his research program for a synthesis of evolutionary and developmental biology. J Exp Zool B Mol Dev Evol. 2006;306(2):89-106.

6. Glushkov VM. Thinking and cybernetics. Russ Stud Philos. 1964;2(4):3-13.
7. Eglash R. African influences in cybernetics. The cyborg handbook. 1995:17-27.
8. Warwick K. Cybernetics: The modern science of systems. Kybernetes. 1994;23(7):76-85.
9. Eglash R. Cybernetics and American youth subculture. Cult Stud. 1998;12(3):382-409.
10. Yang H, Chen F, Aliyu S. Modern software cybernetics: Trends with new cybernetics. J Syst Softw. 2016.
11. Trokhimchuck PP. Problems of complexity in modern cybernetics and computing science and ways of their resolutions. Bulletin of Kherson National Technical University. 2016;3(58):292-296.