Theory and practice in secure software development lifecycle: A comprehensive survey

Martin Otieno 1, David Odera 2,* and Jairus Ekume Ounza 3

1 Jaramogi Oginga Odinga University of Science and Technology, Bondo, Kenya.
2 Tom Mboya University, Homa-Bay, Kenya.
3 Kabarak University, Nakuru, Kenya.

World Journal of Advanced Research and Reviews, 2023, 18(03), 053–078

Publication history: Received on 11 April 2023; revised on 30 May 2023; accepted on 01 June 2023

Article DOI: https://doi.org/10.30574/wjarr.2023.18.3.0944

Abstract

Software development security refers to the practice of integrating security measures and considerations throughout the software development lifecycle to ensure the confidentiality, integrity, and availability of software systems. It involves identifying, mitigating, and eliminating security vulnerabilities and threats that could be exploited by attackers. The goal of this paper is to survey the various concepts and methodologies directed towards software security, and the identification of any missing gaps. Based on the findings, it is noted that the development of secure software requires a proactive and comprehensive approach. It begins with establishing secure design principles and incorporating security requirements from the initial stages of development. Here, secure coding practices, such as input validation, output encoding, and secure authentication and authorization mechanisms, are employed to prevent common security vulnerabilities. In addition, regular security testing, including penetration testing and vulnerability scanning, helps identify and address potential weaknesses in the software. Normally, code reviews and security audits are conducted to ensure adherence to secure coding practices and identify any security flaws. It is important that security training and awareness programs be provided to developers and other stakeholders to foster a security-conscious culture. To minimize potential vulnerabilities, secure configuration management, which involves properly configuring servers, networks, and dependencies may be utilized. On the other hand, regular updates and patching are essential to address known security vulnerabilities in software components. To guide their software development security practices, organizations may follow established security standards and frameworks such as ISO 27001 or NIST Cybersecurity Framework. By prioritizing software development security, organizations can protect sensitive data, prevent unauthorized access, and mitigate the risk of security breaches and incidents. In the long run, this helps build trust with users and stakeholders, enhances the reputation of the software, and reduces the potential impact of security incidents on the organization.

Keywords: Attacks; Coding; Methodologies; Privacy; Security; Software; SDLC

1. Introduction

Software development is the process of creating, designing, coding, testing, and maintaining software systems [1]-[3]. It involves a combination of technical skills, problem-solving abilities, and creative thinking to translate concepts and requirements into functional software applications. As explained in [4], software development plays a fundamental role in today’s digital world, powering various industries, organizations, and everyday activities. The goal is to produce reliable, efficient, and user-friendly software that meets the needs of its intended users. It encompasses a broad range of activities, including understanding user requirements, designing system architectures, writing code, debugging and testing [5], and deploying software to production environments. Effective software development requires collaboration

* Corresponding author: David Odera

Copyright © 2023 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0.
Software security is a critical aspect of modern software development and usage. With the increasing reliance on software systems in various domains, the need to protect these systems from security threats has become paramount [12]-[14]. It focuses on ensuring the confidentiality, integrity, and availability of software and the data it handles. The primary objective of software security is to identify and mitigate vulnerabilities, weaknesses, and threats that could potentially be exploited by attackers [15]. It involves implementing measures, practices, and controls to protect software systems from unauthorized access, data breaches, information leakage, and other security breaches. It encompasses various layers and aspects, including secure coding practices, secure design principles, secure configurations, secure authentication and authorization mechanisms, encryption, access control, secure communication protocols, and robust security testing [16]-[18]. These measures are designed to minimize the risk of security vulnerabilities, such as injection attacks, cross-site scripting (XSS), buffer overflows, and improper data handling. In addition to technical considerations, software security also involves promoting a security-conscious culture within organizations [19]-[23]. This includes raising awareness about security best practices, providing training to developers and users, and establishing processes for security incident response and management. The authors in [24] and [25] point out that it is important to integrate security into the software development lifecycle from the early stages to address security requirements proactively.

The consequences of software security breaches can be severe, ranging from financial losses and reputational damage to legal liabilities and privacy violations. Software security is not a one-time effort but requires continuous monitoring, updating, and adapting to address emerging threats and vulnerabilities [26]-[28]. Organizations must stay vigilant and proactive in managing software security risks throughout the lifecycle of software systems. As technology evolves, so do the techniques and tactics employed by attackers. Therefore, software security is an ongoing challenge that necessitates staying up-to-date with the latest security practices, emerging vulnerabilities, and countermeasures. Collaboration, information sharing, and adherence to industry standards and best practices are essential in ensuring robust software security [29]. According to [30], software security is of utmost importance in today's interconnected world. It involves implementing security measures, adhering to secure coding practices, and maintaining a security-conscious culture to protect software systems from security threats. By prioritizing software security, organizations can safeguard their software assets, protect user data, and maintain trust in an increasingly digital landscape. In this paper, the following contributions are acclaimed:

- The software development lifecycle is examined in great depth in order to establish how it can be made secure.
- The various software development approaches are explored so as to understand how they can shape the development process.
- The concept of software security is discussed and some the key rationale for software security are identified.
- The predominant software security risks are established and the techniques for addressing them described.
- The theory and practice of software security is discussed, including their key aspects.
- The methodologies for identifying and eliminating security vulnerabilities are illustrated, including the tools for identifying and eliminating these security vulnerabilities.
- The various techniques to prove the absence of vulnerabilities are described, including the essential guidelines for building secure software.

The following sub-sections discuss the major issues in software development process in some greater details. Towards the end of this article, research gaps are identified which need immediate attention.
2. The software development lifecycle

The software development lifecycle (SDLC) is a structured and systematic approach to developing software applications [31]. It encompasses a series of phases and activities that guide the development process from initial conception to deployment and maintenance. It plays a crucial role in software development, including software security. According to [32] and [33], SDLC is a structured approach that outlines the steps involved in the development, deployment, and maintenance of software systems. It provides a systematic and structured approach to software development, ensuring that security considerations are integrated at each stage. By following the SDLC, organizations can minimize security risks, address vulnerabilities proactively, and deliver software systems that are more resilient to security threats [34]-[37]. As explained in [38], SDLC serves as a framework to guide developers and teams in building secure software [39] and facilitates the establishment of a security-conscious culture within organizations. While specific SDLC models may vary, the stages in Fig.1 are commonly found in most software development lifecycles:

![Figure 1 Typical SDLC model](image)

At the requirements gathering phase, the project team identifies and documents the software requirements based on user needs, business objectives, and stakeholder input [40]. This involves understanding the purpose of the software, its functionality, and any constraints or dependencies. However, at the analysis and design stage, the project team analyzes the requirements and designs the software architecture, data structures, and user interface [41]. This includes creating system and component-level designs, defining the system's behavior and interactions, and selecting appropriate technologies [42]. During implementation (coding) phase, the software development team translates the design specifications into actual code [43], [44]. They follow coding standards, use appropriate programming languages, and apply software engineering best practices to build the software components and functionality.

As explained in [45] and [46], testing is a critical phase to ensure that the software meets the specified requirements and functions correctly. It involves creating and executing test cases, identifying defects or bugs, and verifying that the software performs as intended. Different testing techniques such as unit testing, integration testing, system testing, and acceptance testing are employed to validate the software [47]. Once the software has passed testing and quality assurance, it is deployed to the production environment or made available to end-users [48]. This phase involves activities such as installation, configuration, and integration with existing systems. After deployment, the software enters the maintenance phase [49]-[51]. This involves monitoring its performance, addressing any issues or bugs discovered in production, and making necessary updates or enhancements. Maintenance may include bug fixes, security patches, performance optimizations [52], and feature additions based on user feedback and changing requirements.

Throughout the SDLC, various project management and development methodologies can be applied, such as waterfall, agile, or DevOps, to ensure effective collaboration, manage project risks, and deliver high-quality software [53], [54]. Additionally, continuous integration and continuous delivery (CI/CD) practices are often implemented to automate builds, testing, and deployment processes, enabling faster and more frequent software releases. It is important to note that the software development lifecycle is not always linear or strictly sequential. Iterative and incremental approaches are commonly used, where feedback from testing and user evaluations informs subsequent iterations of development, allowing for continuous improvement and adaptation.
Software development approaches play a crucial role in shaping how software is planned, developed, and delivered [55]. These approaches provide frameworks, methodologies, and guidelines for organizing and executing the software development process. According to [56], they offer structure, guidance, and best practices for the software development process. In addition, they support project organization, efficiency, collaboration, risk management, quality assurance, and alignment with business goals. Table 1 presents some of the key roles played by the software development approaches.

Table 1 Roles played by software development approaches

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment with business goals</td>
<td>Help align development efforts with business goals and customer needs. They emphasize the value of delivering working software that meets customer expectations and provides business benefits [57]. By focusing on delivering value and addressing customer priorities, software development approaches enable organizations to achieve their strategic objectives.</td>
</tr>
<tr>
<td>Process efficiency and agility</td>
<td>They promote process efficiency [58] and agility. They provide methodologies and techniques for optimizing the development workflow, managing dependencies, and minimizing waste. Agile approaches, such as Scrum or Kanban, emphasize iterative and incremental development, allowing for flexibility, adaptability, and faster delivery of software.</td>
</tr>
<tr>
<td>Project organization and planning</td>
<td>Provide structures for organizing and planning projects. They define roles and responsibilities, establish communication channels, and facilitate coordination among team members. By following a development approach, teams can effectively manage resources, set clear goals and objectives, and establish project timelines [59].</td>
</tr>
<tr>
<td>Quality assurance and testing</td>
<td>Incorporate quality assurance and testing practices into the development process. They provide guidelines for conducting various testing activities, including unit testing, integration testing, and user acceptance testing [60]. These approaches help ensure that software meets quality standards, functions as intended, and is free from bugs and vulnerabilities.</td>
</tr>
<tr>
<td>Continuous improvement and learning</td>
<td>Software development approaches promote continuous improvement and learning [61], [62]. They encourage retrospectives, where teams reflect on their processes, identify areas for improvement, and implement changes to enhance efficiency and quality. Continuous learning and adaptation are core principles in Agile approaches, fostering a culture of continuous improvement and innovation.</td>
</tr>
<tr>
<td>Security and quality assurance</td>
<td>Software development approaches increasingly emphasize security and quality assurance practices. They integrate security considerations, such as secure coding practices, security testing, and vulnerability [63] management, into the development process. These approaches promote a security-conscious mindset and help address security vulnerabilities proactively [64], [65].</td>
</tr>
<tr>
<td>Collaboration and communication</td>
<td>They foster collaboration and effective communication among team members and stakeholders. They provide frameworks for conducting regular meetings, gathering feedback, and aligning the development process with customer needs. Collaboration tools and practices, such as daily stand-up meetings and user story workshops, are often integrated into these approaches to enhance teamwork and stakeholder engagement [66].</td>
</tr>
<tr>
<td>Risk management</td>
<td>Software development approaches address risk management by identifying, assessing, and mitigating risks throughout the development lifecycle [67]. They provide mechanisms for risk identification, risk analysis, and risk mitigation planning. By proactively managing risks, development teams can minimize the impact of potential issues and make informed decisions to mitigate risks effectively [68].</td>
</tr>
</tbody>
</table>

By following a development approach, organizations can improve development outcomes, enhance customer satisfaction, and deliver high-quality software that meets security requirements. According to [69], there are several software development approaches or methodologies that organizations and development teams can adopt to guide their software development process. Table 2 describes some of the commonly used approaches.
Table 2 Common software development approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Particulars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme Programming (XP)</td>
<td>Is an Agile methodology that emphasizes teamwork, communication, and high-quality software [70]. Promotes frequent releases, continuous testing, pair programming, collective code ownership, and customer involvement. Emphasizes simplicity, flexibility, and adaptability in response to changing requirements.</td>
</tr>
<tr>
<td>Spiral</td>
<td>Combines elements of both Waterfall and iterative development [71]. Involve multiple iterations, each consisting of planning, risk analysis, development, and customer evaluation. Incorporates feedback loops and allows for the incorporation of changes and refinements throughout the development process.</td>
</tr>
<tr>
<td>Agile</td>
<td>Agile methodologies, such as Scrum and Kanban, prioritize flexibility, collaboration, and iterative development [72]. Agile teams work in short cycles called sprints, delivering functional increments of software at the end of each sprint. Requirements and solutions evolve through collaboration between cross-functional teams and stakeholders.</td>
</tr>
<tr>
<td>Feature-Driven Development (FDD)</td>
<td>Is an iterative and incremental approach that focuses on delivering specific features. Emphasizes the use of domain modelling, feature identification [73], and short development cycles. Divides the development process into manageable feature-driven workflows and encourages regular feature delivery [74].</td>
</tr>
<tr>
<td>Waterfall</td>
<td>Is a linear and sequential approach to software development [75]. It follows a predefined set of phases: requirements gathering, design, implementation, testing, deployment, and maintenance. Each phase has its specific goals and deliverables, and progress moves in a straightforward, top-down manner.</td>
</tr>
<tr>
<td>DevOps</td>
<td>Is a cultural and technical approach that emphasizes collaboration and integration between development and operations teams [76]. Aims to automate and streamline the software delivery process, allowing for frequent and reliable deployments. Its practices include continuous integration, continuous delivery, infrastructure as code, and automated testing.</td>
</tr>
<tr>
<td>Lean</td>
<td>Focuses on reducing waste, improving efficiency, and continuously delivering value to customers [77]. Emphasizes the elimination of non-value-adding activities, streamlining processes, and optimizing workflow. Lean principles are often combined with Agile methodologies to achieve faster and more efficient [78] software development.</td>
</tr>
<tr>
<td>Rapid Application Development (RAD)</td>
<td>Focuses on rapid prototyping and iterative development. Aims to accelerate the development process by emphasizing user involvement, iterative feedback, and quick delivery of working software [79]. Often involves the use of visual development tools and components to expedite development.</td>
</tr>
</tbody>
</table>

Each of these approaches has its strengths and suitability for different projects and teams. It is therefore important to select the approach that best aligns with project requirements, team dynamics, and organizational goals. In addition,
many organizations also adopt hybrid approaches, combining elements from multiple methodologies to tailor a development process that suits their specific needs.

4. Software security

Software security refers to the practice of protecting computer systems and software applications from unauthorized access, use, disclosure, disruption, modification, or destruction [80]. It involves implementing measures to prevent security vulnerabilities and mitigate the risks associated with potential attacks [81]-[83]. As explained in [84], the role of software security is to protect software systems from security threats, vulnerabilities, and attacks. It involves implementing measures and practices to ensure the confidentiality, integrity, and availability of software and the data it processes. According to [85], software security is an endevour to protect software systems, data, and users from security threats. It encompasses various practices, controls, and processes that enable organizations to develop, deploy, and maintain secure software systems [86]-[88]. By prioritizing software security, organizations can mitigate risks, protect sensitive information, and maintain the trust and confidence of their stakeholders. As discussed in [89], understanding and mitigating software security risks is crucial to protect software systems and the data they handle. This may involve implementing appropriate security controls, conducting regular security assessments, following secure coding practices, and staying updated with security patches and updates. Table 3 describes the key rationale for software security.

Table 3 Rationale for software security

<table>
<thead>
<tr>
<th>Rationale</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building trust and confidence</td>
<td>Software security is essential for building trust and confidence among users, customers, and stakeholders [90]. When software is secure and protected against security threats, it instills confidence in users that their data and information are safe. Strong security measures help organizations establish a positive reputation, retain customer loyalty, and gain a competitive edge in the market.</td>
</tr>
<tr>
<td>Preventing exploitation of vulnerabilities</td>
<td>It focuses on identifying and mitigating vulnerabilities that could be exploited by attackers. This includes addressing coding flaws, insecure configurations, and design weaknesses that can lead to security breaches [91]. By proactively addressing vulnerabilities, software security reduces the risk of unauthorized access, data manipulation, and system compromise.</td>
</tr>
<tr>
<td>Fostering a security-conscious culture</td>
<td>Aims to foster a security-conscious culture within organizations. It involves promoting security awareness, providing training to developers and stakeholders, and encouraging secure coding practices. By integrating security into the software development process, organizations prioritize security from the initial stages and ensure a proactive approach to software security [92].</td>
</tr>
<tr>
<td>Protecting sensitive information</td>
<td>Aims to safeguard sensitive information such as personal data [93], financial records, intellectual property, and customer information. By implementing appropriate security controls, encryption mechanisms, and access management, software security prevents unauthorized access, data breaches, and information leakage [94].</td>
</tr>
<tr>
<td>Mitigating risks and compliance</td>
<td>Plays a vital role in mitigating risks associated with legal and regulatory compliance. Helps organizations meet industry-specific standards, data protection regulations, and privacy requirements [95]. Helps organizations avoid legal liabilities, financial losses, and reputational damage associated with non-compliance.</td>
</tr>
<tr>
<td>Responding to security incidents</td>
<td>Includes incident response and management processes to address security incidents effectively [96]. Involves implementing incident detection mechanisms, establishing incident response plans, and conducting forensic investigations. Helps minimize the impact of breaches, identify root causes, and prevent future occurrences.</td>
</tr>
</tbody>
</table>

58
Ensuring availability and reliability | Aims to ensure the availability and reliability of software systems [97]. By implementing measures such as redundancy, load balancing, and disaster recovery mechanisms, software security helps prevent disruptions caused by denial-of-service attacks [98], system failures, or other malicious activities. Ensures that software systems remain operational and accessible to legitimate users.

By addressing software security risks, organizations can reduce the likelihood and impact of security breaches and safeguard their software assets. Some of the key aspects considered in software security include threat modeling [99]-[102], secure design and development [103], authentication and authorization [104], data protection [105], regular updates and patching [106], secure configuration and deployment, secure testing, incident response and recovery, user education and awareness, compliance and regulations.

In threat modeling, the goal is to analyze and understand the potential threats and vulnerabilities that could affect the software [107]. This involves identifying potential attackers, their motivations, and the potential risks to the system. However, for secure design and development, the aim is to follow secure coding practices, such as input validation, output encoding, and proper error handling [108]. This helps prevent common vulnerabilities like injection attacks such as SQL injection [109] and cross-site scripting. On the other hand, implementing strong authentication mechanisms, such as multi-factor authentication can ensure that only authorized users can access the software [110]. Additionally, implementing proper authorization controls ensures that users have appropriate permissions and access privileges [111]-[114]. For data protection, sensitive data is protected both at rest (stored) and in transit (being transmitted over a network). Basically, encryption techniques, such as using strong algorithms and secure protocols like SSL/TLS, help safeguard data from unauthorized access [115]. On the other hand, keeping software up to date with the latest security patches and updates is crucial. This includes both the software itself and any libraries or dependencies it relies on. Regularly monitoring for security vulnerabilities and promptly applying patches helps mitigate known risks [116].

Proper configuration of software and infrastructure is essential for security. This means that default configurations should be changed, unnecessary features and services should be disabled, and access controls should be properly defined [117]. On the other hand, conducting thorough security testing, such as penetration testing and vulnerability scanning, helps identify weaknesses and vulnerabilities in the software [118]. Essentially, regular testing, including both manual and automated techniques, allows for proactive identification and mitigation of security risks. As explained in [119], having a well-defined incident response plan in place enables the organization to respond promptly and effectively to security incidents. It includes procedures for detecting, responding to, and recovering from security breaches [120]. On the other hand, educating users about good security practices, such as strong password management, recognizing phishing attempts, and exercising caution while using the software, helps create a security-conscious environment [121]. Researchers in [122] explain that depending on the industry and geographical location, software may need to adhere to specific compliance standards and regulations such as GDPR, HIPAA. It is important to understand and implement the necessary security controls to meet these requirements.

Based on the above discussion, it is clear that software security is an ongoing process that requires continuous monitoring, adaptation, and improvement. By incorporating security practices throughout the software development lifecycle, organizations can mitigate risks and protect their systems, data, and users from potential threats.

### 5. Common software security risks

Software security risks refer to potential vulnerabilities, threats, and weaknesses that can compromise the security of software systems [123]. These risks pose a threat to the confidentiality, integrity, and availability of the software, as well as the data it processes. These risks can lead to unauthorized access, data breaches, system compromise, financial loss, reputational damage, and legal liabilities. Table 4 discusses some of common software security risks.

According to [135], software security risks refer to the vulnerabilities and weaknesses that can be exploited by attackers to compromise the security of software applications. Therefore, understanding and addressing these risks is crucial to protect sensitive data, maintain system integrity, and ensure the confidentiality and availability of software resources. Threats such as injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Insecure Direct Object References (IDOR), security mis-configurations, weak authentication and authorization, Denial-of-Service (DoS), vulnerabilities in third-party libraries, insider threats and lack of secure communication all present some levels of risks to the software systems [136],[137].
Table 4 Common software security risks

<table>
<thead>
<tr>
<th>Software risk</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Third-party dependencies</td>
<td>Relying on third-party libraries, frameworks, or components introduces the risk of inheriting vulnerabilities from these dependencies [124]. If the third-party software has security flaws or is not regularly updated, it can create security risks within the software system.</td>
</tr>
<tr>
<td>Insufficient logging and monitoring</td>
<td>Inadequate logging and monitoring capabilities can hinder the detection and response to security incidents [125]. Without proper logging, it becomes challenging to identify suspicious activities, track unauthorized access, or investigate security breaches effectively.</td>
</tr>
<tr>
<td>Software vulnerabilities</td>
<td>Are weaknesses in the software’s code or design that can be exploited by attackers [126]. These vulnerabilities can include buffer overflows, injection attacks, cross-site scripting (XSS), and insecure direct object references [127]. Exploiting these vulnerabilities can lead to data breaches, code execution, or unauthorized system manipulation.</td>
</tr>
<tr>
<td>Insecure authentication and authorization</td>
<td>Weak or inadequate authentication and authorization mechanisms can allow unauthorized users to gain access to the software system and its resources [128]. This can result in unauthorized data access, privilege escalation, and unauthorized actions within the system.</td>
</tr>
<tr>
<td>Insecure data storage and transmission</td>
<td>Insecure handling of sensitive data, such as storing data in plain text or transmitting data over unsecured channels, can expose data to unauthorized access or interception [130]. This can result in data breaches, identity theft, and compromise of confidential information.</td>
</tr>
<tr>
<td>Inadequate input validation</td>
<td>Insufficient input validation can lead to various security risks, such as injection attacks [131], including SQL injection and command injection. Without proper input validation, malicious inputs can manipulate the software’s behavior, leading to unauthorized access, data corruption, or system compromise [132].</td>
</tr>
<tr>
<td>Lack of secure configuration</td>
<td>Incorrect or insecure configurations of software components, servers, and network devices can create security vulnerabilities [133]. Default passwords, mis-configured access controls, and unnecessary services or ports can provide avenues for attackers to exploit and gain unauthorized access to the system.</td>
</tr>
<tr>
<td>Social engineering attacks</td>
<td>Software security risks also include social engineering attacks, where attackers manipulate individuals to gain unauthorized access or divulge sensitive information [134]. Phishing, pre-texting, and baiting are examples of social engineering techniques that can exploit human vulnerabilities to breach software security.</td>
</tr>
</tbody>
</table>

According to [138], injection attacks, such as SQL injection and cross-site scripting (XSS), involve malicious code or commands being injected into software components that interpret user input. These attacks can lead to unauthorized data access, data manipulation, or code execution. In particular, XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users, potentially compromising their browsers and enabling attacks like session hijacking or phishing. On the other hand, CSRF attacks trick authenticated users into performing unintended actions on a website by leveraging their trusted session [139]. This can result in unauthorized changes to user settings or data. As explained in [140], IDOR vulnerabilities occur when an application exposes direct references to internal objects or resources, allowing attackers to manipulate object identifiers and gain unauthorized access to sensitive data or functionalities [141]. On the other hand, security mis-configurations arise from improperly configured software components, servers, databases, or frameworks. This can leave software systems vulnerable to attacks, such as unauthorized access or privilege escalation. The researchers in [142] and [143] point out that weak authentication mechanisms, such as the use of weak passwords or insufficient authentication protocols, can lead to unauthorized access [144]. Inadequate authorization controls may allow unauthorized users to gain elevated privileges or access sensitive functionalities [145], [146]. On the other hand, DoS attacks aim to disrupt the availability of a service or system by overwhelming it with excessive requests or exploiting vulnerabilities that exhaust system resources. These attacks
can render a system unusable, leading to service disruptions or downtime. As discussed in [147], many software applications rely on third-party libraries or components, which may have vulnerabilities. If these vulnerabilities are not properly addressed or updated, they can introduce security risks into the software and be exploited by attackers [148], [149]. On the other hand, insider threats involve individuals with authorized access to systems or information who misuse their privileges for malicious purposes or inadvertently introduce security risks [130]. These threats can come from employees, contractors, or anyone with trusted access to the system. Researchers in [151] point out that inadequate encryption and insecure communication protocols can expose sensitive data to interception or unauthorized access during transmission, compromising data confidentiality and integrity.

Evidently, addressing the above software security risks requires implementing security best practices throughout the software development lifecycle, conducting regular security assessments and testing, and staying informed about emerging vulnerabilities and mitigation strategies. It is essential to prioritize security and adopt a proactive approach to protect software systems and the data they handle. The following section explores the role of theory and practice towards achieving this protection.

6. Theory and practice of software security

This encompasses both the conceptual understanding and practical implementation of security measures in software development and maintenance. It also involves various principles, models, methodologies, and techniques aimed at designing, building, and maintaining secure software systems [152]. Table 5 illustrates some of the key aspects of the theory and practice of software security.

<table>
<thead>
<tr>
<th>Table 5 Key aspects of the theory and practice of software security</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aspect</strong></td>
</tr>
<tr>
<td>Security Assurance</td>
</tr>
<tr>
<td>Secure Development Lifecycle (SDL)</td>
</tr>
<tr>
<td>Secure Design Patterns</td>
</tr>
<tr>
<td>Secure Coding Practices</td>
</tr>
<tr>
<td>Security Principles</td>
</tr>
<tr>
<td>Threat Modelling</td>
</tr>
</tbody>
</table>
Secure Software Maintenance

Software security is not a one-time effort; it requires ongoing maintenance and updates. Patch management, vulnerability management, and timely software updates are crucial to address newly discovered vulnerabilities and security threats [160].

Security Culture and Training

Building a security-conscious culture within an organization is essential. Involves creating awareness, providing training, and promoting security best practices among developers, testers, and other stakeholders involved in the software development process [161].

Security Testing

Aims to identify vulnerabilities and weaknesses in software systems. Includes techniques such as penetration testing, vulnerability scanning, code reviews, and security-focused testing methodologies [162]. By uncovering vulnerabilities, organizations can address them before deploying the software.

Security Governance

Establishes the framework and processes for managing software security. Involves defining security policies, assigning responsibilities, implementing security controls, and ensuring compliance with applicable regulations and standards [163], [164]. Provides the structure and accountability necessary to manage and maintain software security effectively.

It is important to note that the theory and practice of software security are continually evolving due to the evolving threat landscape. As such, keeping up with emerging security trends, staying informed about new vulnerabilities, and adapting security practices accordingly is vital to ensuring the resilience and security of software systems.

7. Methodologies for identifying and eliminating security vulnerabilities

Identifying and eliminating security vulnerabilities in software applications is a critical process to protect against potential attacks and safeguard sensitive data [165]. There are several methodologies and approaches for identifying and eliminating security vulnerabilities in software applications. These include threat modeling, secure coding practices, secure SDLC, security code reviews, penetration testing, security scanning and vulnerability assessment. According to [166], threat modeling is a proactive approach to identify potential security threats and vulnerabilities in the early stages of software development. It involves analyzing the system architecture, components, and interactions to understand the potential attack vectors. By identifying threats and their potential impact, developers can prioritize security measures and design the software with security in mind [167]. To uphold secure coding practices, there is need to adhering to secure coding practices. This helps prevent common vulnerabilities and strengthens the overall security of the software. Practices such as input validation, output encoding, secure error handling, proper session management, and secure communication protocols can significantly reduce the risk of vulnerabilities like injection attacks, cross-site scripting (XSS), and others. According to [168], performing regular code reviews specifically focused on security can help identify vulnerabilities in the codebase. These code reviews involve manual examination of the code to detect insecure coding patterns, potential vulnerabilities, and adherence to secure coding practices. This can be done by experienced developers or security professionals to ensure a thorough assessment.

Penetration testing, also known as ethical hacking, involves simulating real-world attacks on the software to identify vulnerabilities [169]. Here, skilled security professionals use various tools and techniques to probe the software's security defenses, exploit vulnerabilities, and provide recommendations for remediation. This helps identify weaknesses that may not be apparent through other security assessment methods. On the other hand, automated security scanning tools and vulnerability assessment scanners can be used to detect common vulnerabilities in software applications [170]. These tools scan the application’s code, configurations, and dependencies to identify potential weaknesses. In so doing, they help uncover issues such as insecure configurations, outdated libraries, and known vulnerabilities, providing a starting point for remediation efforts [171]. As explained in [172], educating developers, testers, and other stakeholders about secure coding practices, common vulnerabilities, and emerging threats is essential. Security training programs increase awareness and provide the knowledge necessary to develop and maintain secure software applications. It helps in cultivating a security-conscious culture within the organization. Similarly, incorporating security activities at each phase of the SDLC, such as threat modeling, secure design, secure coding, security testing, and security-focused quality assurance. A well-defined and consistently followed SDLC ensures that security is a fundamental aspect of the development process.
As pointed out in [174], these methodologies are not mutually exclusive, and a combination of approaches is often necessary to achieve comprehensive vulnerability identification and remediation. Additionally, staying updated with security advisories, industry best practices, and security communities can provide valuable insights into emerging vulnerabilities and mitigation strategies.

8. Tools for identifying and eliminating security vulnerabilities

There are numerous tools available that can assist in identifying and eliminating security vulnerabilities in software applications [175]. They automate certain processes, provide analysis, and help identify potential weaknesses. Table 6 presents some of these tools.

**Table 6 Security vulnerabilities identification and elimination tools**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Composition Analysis (SCA) Tools</td>
<td><em>Black Duck</em>: Scans application dependencies and identifies known vulnerabilities in open-source libraries and components.</td>
</tr>
<tr>
<td></td>
<td><em>Sonatype Nexus Lifecycle</em>: Helps identify and manage open-source components in software development, ensuring they are free from vulnerabilities and licensing issues.</td>
</tr>
<tr>
<td></td>
<td><em>WhiteSource</em>: Offers SCA capabilities to identify and manage open-source components and libraries used in software applications.</td>
</tr>
<tr>
<td>Static Application Security Testing (SAST) Tools</td>
<td><em>Fortify</em>: A popular SAST tool that analyzes source code to identify security vulnerabilities and coding errors.</td>
</tr>
<tr>
<td></td>
<td><em>Veracode</em>: Offers SAST capabilities for identifying flaws in code and providing recommendations for remediation.</td>
</tr>
<tr>
<td></td>
<td><em>Checkmarx</em>: Provides static code analysis to identify security vulnerabilities and offers integration with various development environments.</td>
</tr>
<tr>
<td></td>
<td><em>Seeker</em>: Offers IAST capabilities for detecting and verifying security vulnerabilities in web applications.</td>
</tr>
<tr>
<td>Security Information and Event Management (SIEM) Tools</td>
<td><em>Splunk</em>: A SIEM tool that collects and analyzes log data from various sources to detect security incidents and anomalies.</td>
</tr>
<tr>
<td></td>
<td><em>Elastic Stack</em>: An open-source toolset that includes Elasticsearch, Logstash, and Kibana for log management, analysis, and visualization.</td>
</tr>
<tr>
<td>Security Scanners and Analyzers</td>
<td><em>Qualys</em>: Offers cloud-based vulnerability management and scanning for identifying and prioritizing vulnerabilities in systems and applications.</td>
</tr>
<tr>
<td></td>
<td><em>Rapid7 AppSpider</em>: Provides automated scanning for web applications to identify security vulnerabilities and misconfigurations.</td>
</tr>
<tr>
<td>Dynamic Application Security Testing (DAST) Tools</td>
<td><em>Burp Suite</em>: A comprehensive DAST tool that allows manual and automated security testing of web applications.</td>
</tr>
<tr>
<td></td>
<td><em>OWASP ZAP</em>: An open-source DAST tool that helps identify common web application vulnerabilities and supports automation.</td>
</tr>
<tr>
<td></td>
<td><em>Acunetix</em>: Provides DAST capabilities to scan web applications for security vulnerabilities, including injection attacks, XSS, and more.</td>
</tr>
<tr>
<td>Penetration Testing Tools</td>
<td><em>Metasploit</em>: A popular penetration testing framework that provides a range of tools for exploiting vulnerabilities and assessing security.</td>
</tr>
<tr>
<td></td>
<td><em>Nmap</em>: A versatile network scanning tool that helps identify open ports, services, and potential vulnerabilities in a network.</td>
</tr>
<tr>
<td></td>
<td><em>Nessus</em>: A vulnerability scanner that detects security issues in networks, systems, and applications.</td>
</tr>
</tbody>
</table>
The tools in Table 6 can be integrated into the software development lifecycle and used in combination to provide comprehensive vulnerability identification and remediation. While these tools can assist in vulnerability management, human expertise and analysis are still crucial to interpret results, prioritize vulnerabilities, and apply appropriate remediation measures.

9. Techniques to prove the absence of vulnerabilities

The procedures for proving the absence of vulnerabilities in software applications is a challenging task since it is nearly impossible to guarantee complete absence [176]. However, there are several techniques and practices that can help increase confidence in the security of a system. Some of these approaches include security testing, secure coding practices, code reviews, and security audits, threat modeling, security by design, compliance with security standards, security verification [177] standard, independent security assessments, security awareness and training, secure development frameworks and libraries. According to [178], security testing involves the conducting of comprehensive security testing using techniques such as penetration testing, vulnerability scanning, and code review. These assessments aim to identify and address security vulnerabilities proactively. By performing rigorous testing, organizations can minimize the chances of undetected vulnerabilities. However, secure coding practices encompasses the adoption of security from the initial stages of development. This includes adhering to secure coding guidelines and best practices, implementing input validation, output encoding, proper error handling, and secure communication protocols. By following these practices, developers can minimize the likelihood of introducing vulnerabilities during the coding process. It is important that regular code reviews and security audits be carried out to identify potential vulnerabilities [179]. Experienced developers or security experts can review the codebase to identify coding flaws, design weaknesses, and implementation errors. This helps uncover and address vulnerabilities before they are deployed. In addition, threat modeling techniques can be applied to identify potential security risks and vulnerabilities. Analyze the system architecture, identify potential threats, and assess the impact of those threats. By proactively addressing security risks during the design phase, organizations can minimize the likelihood of introducing vulnerabilities.

As pointed out in [180], a security-by-design approach need to be adopted throughout the software development lifecycle. This involves integrating security considerations from the early stages of development, incorporating secure design principles, and continuously assessing and mitigating potential risks. By making security an inherent part of the development process, the chances of introducing vulnerabilities are reduced. It is also important to comply with established security standards and frameworks such as ISO 27001, NIST Cybersecurity Framework, or OWASP Application Security Verification Standard [181]. Adhering to these standards ensures that security practices and controls are in place, reducing the likelihood of vulnerabilities. Moreover, organizations need to engage third-party security experts or external auditors to perform independent security assessments of the software application [182]. These professionals can evaluate the system for vulnerabilities [183] and provide objective feedback and recommendations for improvement. For enhanced security, organizations need to foster a culture of security awareness among developers, testers, and other stakeholders involved in the software development process. As explained in [184], regular training sessions on secure coding practices, emerging threats, and security policies help individuals understand their role in maintaining secure software and reduce the likelihood of introducing vulnerabilities. As explained in [185], it is crucial to leverage established secure development frameworks and libraries that have undergone extensive security testing and auditing. These frameworks and libraries are built with security in mind and can help minimize vulnerabilities resulting from incorrect implementations.

While these techniques can help reduce vulnerabilities, it is important to understand that proving the absolute absence of vulnerabilities is challenging. Security is an ongoing effort, and organizations should continuously monitor, assess, and improve the security posture of their software applications.

10. Essential guidelines for building secure software

To avoid security holes in new software, it is crucial to incorporate security practices throughout the development lifecycle. Some key ways to mitigate the risk of security vulnerabilities include secure design and secure coding practices [186]. In secure design, developers begin with a strong and secure design phase, considering security requirements, threat modeling, and risk analysis during the design process. Identify potential security vulnerabilities and design the system architecture and components to minimize those risks. However, in secure coding practices, developers need to adhere to validate and sanitize user input, using parameterized queries or prepared statements to prevent injection attacks, implementing proper access controls and authentication mechanisms, and securely handling sensitive data [187], [188]. Apply security-focused coding guidelines and standards.
When building secure software, it is crucial to follow essential guidelines and best practices to minimize the risk of security vulnerabilities. Table 7 presents some of these key guidelines to consider.

### Table 7 Guidelines for building secure software

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement Secure Authentication and Authorization</td>
<td>Use strong and properly implemented authentication mechanisms [189]. Enforce strong password policies and consider implementing multi-factor authentication. Implement proper authorization controls to ensure users have appropriate access privileges.</td>
</tr>
<tr>
<td>Validate and Sanitize User Input</td>
<td>Validate and sanitize all user input to prevent common vulnerabilities such as injection attacks (such as SQL injection, XSS). Implement input validation on both the client and server sides to ensure data integrity [190].</td>
</tr>
<tr>
<td>Protect Sensitive Data</td>
<td>Apply encryption algorithms to protect sensitive data, both in transit and at rest [191]. Use appropriate cryptographic protocols and algorithms, and ensure proper key management.</td>
</tr>
<tr>
<td>Follow Secure Coding Practices</td>
<td>Adhere to secure coding practices and coding guidelines that promote security, such as input validation, output encoding, secure error handling, and secure communication protocols [192], [193]. Avoid insecure coding patterns and practices, such as using deprecated functions or insecure configurations.</td>
</tr>
<tr>
<td>Manage Session Security</td>
<td>Implement secure session management techniques, including secure session tokens, session expiration, and secure cookie management [194]. Avoid storing sensitive information in session variables or cookies.</td>
</tr>
<tr>
<td>Handle Errors Securely</td>
<td>Implement proper error handling mechanisms to avoid exposing sensitive information to potential attackers [195]. Provide informative error messages to users without revealing system details that could be exploited.</td>
</tr>
<tr>
<td>Secure Configuration Management</td>
<td>Securely configure servers, frameworks, libraries, and dependencies [196]. Follow security best practices for server configurations, network settings, and access controls. Disable unnecessary services, ports, and protocols to minimize the attack surface.</td>
</tr>
<tr>
<td>Regularly Update and Patch Software</td>
<td>Keep all software components, frameworks, libraries, and dependencies up to date with the latest security patches and updates [197]. Stay informed about security advisories and vulnerabilities [198] related to the software used and promptly apply patches.</td>
</tr>
<tr>
<td>Conduct Security Testing</td>
<td>Perform regular security testing, including penetration testing, vulnerability scanning, and code review [199]. Test for common vulnerabilities such as injection flaws, XSS, security mis-configurations, and access control issues. Utilize both automated tools and manual testing techniques.</td>
</tr>
<tr>
<td>Foster a Security-Conscious Culture</td>
<td>Educate and train developers, testers, and other stakeholders on secure coding practices, emerging threats, and security policies [200]. Promote a culture of security awareness and encourage everyone to take responsibility for building secure software.</td>
</tr>
<tr>
<td>Follow Secure Development Lifecycle (SDLC)</td>
<td>Incorporate security practices into the entire software development lifecycle, from requirements gathering to deployment and maintenance [201]. Include security reviews, testing, and risk assessments at each stage of the SDLC.</td>
</tr>
</tbody>
</table>
Stay Informed about Security

Keep up to date with the latest security practices, standards, and emerging threats [202]. Monitor security communities, subscribe to security mailing lists, and stay informed about security news and developments.

By following these guidelines, organizations can build software with a strong security foundation and reduce the risk of security vulnerabilities [203]. It is important to note that security is an ongoing process, and continuous monitoring, testing, and improvement are necessary to address evolving threats and maintain the security of the software over time.

11. Research gaps

While significant progress has been made in the field of software security, there are still several research gaps that merit further investigation. Some of the key research gaps in software security include:

Secure Software Development Processes: There is a need for more research on effective methodologies, frameworks, and tools for integrating security into the entire software development lifecycle [204]. This includes identifying best practices for secure requirements engineering, secure design, secure coding, and secure testing as shown in Fig.2.

Automated Vulnerability Detection: While there are various automated tools available for vulnerability detection, there is room for improvement in their accuracy, coverage, and effectiveness. Research is needed to develop advanced techniques for automated vulnerability detection, including static analysis, dynamic analysis, and hybrid approaches [205].

Security of Emerging Technologies: With the emergence of new technologies such as Internet of Things (IoT), cloud computing, blockchain, and artificial intelligence (AI), there is a need to explore their unique security challenges and develop robust security solutions and best practices specific to these domains [206].

Secure Software Architectures: Research is needed to explore secure software architectures that can effectively protect against modern threats and vulnerabilities [207], [208]. This includes designing architectures that provide strong isolation, access control mechanisms, secure communication channels, and resilience against attacks.

Secure DevOps and Continuous Security: As organizations increasingly adopt DevOps practices, there is a need for research on integrating security seamlessly into DevOps processes [209]. This involves developing methodologies and tools for continuous security testing, vulnerability management, and secure deployment pipelines.

Human Factors in Software Security: Human factors, such as user behavior, social engineering, and organizational culture, play a significant role in software security [210]. Further research is needed to understand and address human vulnerabilities, improve user awareness and education, and develop effective security training and awareness programs.
Threat Intelligence and Information Sharing: Research is needed to improve the collection, analysis, and sharing of threat intelligence among organizations [211], [212]. This includes developing techniques for automated threat intelligence gathering, effective sharing platforms, and privacy-preserving mechanisms for information exchange.

Secure Software Updates and Patch Management: Software updates and patch management are critical for addressing vulnerabilities [213]-[215]. Research is required to develop efficient and secure mechanisms for software updates, including strategies for safe and reliable patch distribution and installation.

Evaluating Security Controls: There is a need for research on evaluating the effectiveness and impact of security controls and countermeasures [216], [217]. This includes developing metrics, frameworks, and methodologies to measure the security posture of software systems and assess the effectiveness of security controls.

Socio-Technical Aspects of Software Security: Understanding the socio-technical aspects of software security is crucial, as it involves the interaction between technology, people, and organizations [218]-[223]. Research is needed to explore the socio-cultural, economic, and legal factors that influence software security and develop strategies to address them effectively.

Addressing these research gaps will contribute to the advancement of software security, leading to the development of more secure software systems and better protection against evolving threats and vulnerabilities.

12. Conclusion
Software security is of paramount importance in today’s digital landscape where cyber threats are prevalent. Developing secure software requires a holistic and proactive approach that encompasses all stages of the software development lifecycle. By integrating security practices from the initial design phase, employing secure coding practices, conducting regular security testing, and following secure configuration and patching processes, organizations can significantly mitigate the risk of security vulnerabilities. Investing in software security not only protects sensitive data and intellectual property but also safeguards the reputation and trust of the organization. It helps prevent financial losses, legal liabilities, and potential disruptions caused by security incidents. Moreover, prioritizing software security demonstrates a commitment to customer privacy and compliance with regulatory requirements. However, it is important to acknowledge that software security is an ongoing process. Threats and vulnerabilities evolve continuously, and new attack vectors emerge. Organizations should stay vigilant, keep abreast of the latest security practices and technologies, and continuously assess and improve the security posture of their software applications. Ultimately, software security is a shared responsibility among developers, testers, security professionals, and stakeholders. Collaboration, education, and awareness are key in fostering a security-conscious culture and ensuring that security is ingrained in every aspect of the software development process. By doing so, organizations can build robust, resilient, and trusted software systems that protect against threats and instill confidence in users and stakeholders.

Compliance with ethical standards

Acknowledgments
We would like to thank everyone who offered support during the development of this paper.

Disclosure of conflict of interest
The authors declare that they do not have any conflict of interest.

References


[38] Dodson, D., Souppaya, M., & Scarfone, K. (2020). Mitigating the risk of software vulnerabilities by adopting a secure software development framework (ssdf). NIST: Gaithersburg, MD, USA.


